DOWNLOAD PDF LEARNING OBJECTIVE C A PRIMER APPLE

Chapter 1 : Keep Learning Objective-C or move straighâ€¦ - Apple Community

Objective-C is the primary programming language you use when writing software for OS X and iOS. It's a superset of the C programming language and provides object-oriented capabilities and a dynamic runtime.

In August , I took up the torch again. Within one month I was able to complete my first app. Much has changed in three years. While most of what I wrote below still stands, ignore the book recommendations below. Instead, I strongly urge you to buy the two books below. Had they existed three years ago, I would have been able to develop my first app much quicker. I purchased them in August and had my first app done in less than a month: This book assumes you have some programming experience. This book assumes you know nothing about programming at all! If you know nothing about programming at all, start here. Then move on to the iOS Programming book above. I used it as a primer. It was very handy to get me back up to speed and as a reference book as I plowed through the iOS Programming book above. Buy it on Amazon here â€" or get the Kindle version like I did here. April] How hard could it be, I asked myself. I develop and maintain websites and blogs from my home server. How hard could it be to develop a small application for the iPhone? I could whip it up in C or PHP in about a day. I regard the iPhone App Store as a revolutionary new idea that pries control of mobile device apps from the big-bad telco giants and puts it in the hands of average consumers and developers â€" where it belongs. I see cloud computing as a very important part of our collective computing future. I want to get in on the ground floor. If my first simple program works out, I want to develop an iPhone app to work with my wishhh. After that, who knows. Any Mac released since , laptop or desktop, should work. Not so much for the rest of us. Learning to Use the Mac and Leopard: Admittedly this will not be a hurdle for most iPhone App developers. It was for me. I have never used an Apple computer of any kind for more than a few minutes â€" ever! Like everything else on the Mac, installing it as my first Mac App was a bit of a challenge. I had never heard of. With the help of this Mac DMG files are Disk Images explanation, that challenge was soon overcome. For 15 years I have been steadfastly opposed to learning object oriented programming "OOP". C was good enough for me. The only language available to develop iPhone apps is Objective-C , as the name implies, an object, oriented superset of the C programming language. The specific part of the framework used for iPhone app development is called Cocoa Touch. With that text under my belt, I began to make significant progress. In the initial days, iPhone developers were islands unto themselves. There was very little support anywhere, whether online tutorials, forums, texts etc. Apple lifted its NDA restrictions on October 1, As a result, there are now a plethora of iPhone application development books coming out â€" See here on Amazon. The first book below is not, however, iPhone specific, but was critically important for me to read before I was able to progress through the next two. It filled in most of the blanks I was missing. You will NOT regret making this purchase. Reading and practicing on the iMac with the example programs contained in the first 8 or so chapters gave me the Objective-C grounding I needed to move forward to the books mentioned below. As of Dec 16, , I believe I have a solid enough grounding to move on to the other two books. Demo iPhone apps project files that correspond to the chapters of the book can be found here. On November 16, , after getting a couple chapters in, I stopped and sought out the Aaron Hillegass text above. Much of it useful. Much of it not especially for newbs like me. I strongly recommend the texts listed above. Once you have a grounding in the basics, the official Apple documents become much more useful. Object-Oriented Programming with Objective-C: Thank goodness for this document. After a few days I was almost ready to give up until I found this guide. It is written for someone who understand C and needs a primer on object oriented programming. This article explained OOP to me in a way that finally made sense to me. The light-bulb went off when I read the analogy between objects to C structures. If you understand C but not object oriented programming, this document can help you over the hump. Introduction to Cocoa Application Tutorial: The next most critical piece to my learning was this Cocoa tutorial. I had initially jumped straight to the various iPhone templates, sample code and tutorials but found myself a bit baffled by them. I decided to first step back and use this tutorial to learn how to build a Mac App using the SDK tools before ploughing ahead with my first test iPhone Apps. It was worth it. It nicely walks you through the MVC

model, view, controller design paradigm assumed in all the other iPhone App documentation see also MVC on Wikipedia , and holds your hand, step by step, through the process of designing and coding your first Cocoa application. I had read this document at the very beginning of my learning process and was somewhat flummoxed by it. Coming back to it again after digesting the OOP with Objective C document and the Cocoa tutorial each above , it made so much more sense and is a ultimately a terrific primer once you have a little learning under your belt. Outside of the iPhone Dev Center, I found the following useful: If there is anything that will delight you in the iPhone App development process its xCode. It just does what you want a development environment to do. So many of the menial tasks of the past are handled effortlessly for you. I am continually delighted as I use this tool with its deep feature set. For details see the xCode Workspace Guide. It took me awhile to figure this beast out. But this too will come. The Interface Builder user guide is here again, available only to those who have registered. One of the last pieces of the iPhone App development puzzle before you start testing on an actual device that is is the iPhone Simulator pictured above. At any time during the development process you can build and test a version of your iPhone App and run it in the iPhone Simulator. There are several online tutorials that hold your hand, step by step, through basic iPhone application development: This includes a terrific step-by-step description of the basic bootstrap process. I wish I had read this one a lot earlier than I did. You will need to be logged into the iPhone Development Center to access this. A computer science student named Brandon at the University of New Mexico has been posting useful iPhone App development tutorials on his icodeblog blog. His forum allows would-be iPhone developers to discuss his tutorials and learn from each other. His tutorials have been invaluable to my learning process.

Chapter 2 : iPhone - Apple Developer

Welcome, Eduardo Resende (log out) Learning Objective-C: A Primer The Objective-C language is a simple computer language designed to enable sophisticated object-oriented programming. Objective-C extends the standard ANSI C language by providing syntax for defining classes, methods, and properties, as well as other constructs that promote.

An instance method is a method whose execution is scoped to a particular instance of the class. In other words, before you call an instance method, you must first create an instance of the class. Class methods, by comparison, do not require you to create an instance, but more on that later. Figure 2 shows the declaration of the insertObject: The declaration is preceded by a minus - sign, which indicates that this is an instance method. The colon characters declare the presence of a parameter. If a method has no parameters, you omit the colon after the first and only signature keyword. In this example, the method takes two parameters. The message in this case is the method signature, along with the parameter information the method needs. All messages you send to an object are dispatched dynamically, thus facilitating the polymorphism behavior of Objective-C classes. In other words, if a subclass defines a method with the same signature as one of its parent classes, the subclass receives the message first and can opt to forward the message or not to its parent. Inside the brackets, the object receiving the message is on the left side and the message along with any parameters required by the message is on the right. For example, to send the insertObject: The return value from each nested message is used as a parameter, or as the target, of another message. For example, you could replace any of the variables used in the previous example with messages to retrieve the values. Thus, if you had another object called myAppObject that had methods for accessing the array object and the object to insert into the array, you could write the preceding example to look something like the following: When messaging a class, the method you specify must be defined as a class method instead of an instance method. The syntax for a class method declaration is identical to that of an instance method, with one exception. In this case, the arrayWithCapacity: Properties do not create new instance variables in your class declaration. They are simply a shorthand for defining methods that access existing instance variables. Classes that expose instance variables can do so using the property notation instead of using getter and setter syntax. Because most accessor methods are implemented in similar ways, properties eliminate the need to provide a distinct getter and setter method for each instance variable exposed in the class. Instead, you specify the behavior you want using the property declaration and then synthesize actual getter and setter methods based on that declaration at compile time. The basic definition uses the property compiler directive, followed by the type information and name of the property. You can also configure the property with custom options, which define how the accessor methods behave. The following example shows a few simple property declarations:

Chapter 3 : iPhone App Development â€“ Where to Start â€“ Daleisphere

Learning Objective-C: A Primer The Objective-C language is a simple computer language designed to enable sophisticated object-oriented programming.

If you want to be an iOS developer, you will still need to know Objective-C. Objective-C is easier to learn than Swift. Once you know Objective-C, it will be easy to learn Swift. Before I proceed, let me preface this with a confession of love for Swift. The syntax is lovely. The enum construct is gorgeous. Swift can call C functions, but I believe that if you are working with a lot of C functions and types, you will want to code in Objective-C. The community talks in Objective-C. Objective-C is the language we have used for the last six years to describe to each other how the Cocoa Touch libraries work. The frameworks are written Objective-C. If you want to understand what the debugger is telling you, you will need to understand Objective-C. Objective-C is stable and well-tested. Swift looks great, but the language is evolving and the compiler is immature. If I were making a significant bet on developing an app this year, I would still use Objective-C. Objective-C is easier to learn than Swift C is a really simple little language, and Objective-C is a really simple little extension to C. Swift has many rules that Objective-C does not. I, as an instructor, am already trying to figure out how I will explain the rules around optional variables and the proper use of? These extra rules mean that the compiler can be much more pedantic about enforcing good coding practices, but it also means that the language will take longer to learn. Objective-C requires programmers to be explicit. The Swift language lets the compiler do more work for the programmer. This is greatâ€”less typing for the programmer, right? Explicit languages are easier for beginners to understand. For example, generics make type checking better in Swift, but it makes that language considerably more complex. The difficult ideas that drive Objective-C like objects, strong and weak references, and inheritance are exactly the same in Swiftâ€”they are just expressed using a different syntax.

Chapter 4 : Learning Objective-C: A Primer - çˆ±ç¨‹åº•ç½‘

as-apple-retains-revenue-lead/ iOS Application Compiler Architecture. C++. Objective-C. Java Virtual 10 blog.quintoapp.com Source.

The basic syntax for calling a method on an object is this: Methods can return a value: All Objective-C object variables are pointers types. In many languages, nested method or function calls look like this: In Objective-C, nested messages look like this: Multi-Input Methods Some methods take multiple input values. In Objective-C, a method name can be split up into several segments. In the header, a multi-input method looks like this: You call the method like this: NO]; These are not just named arguments. The method name is actually writeToFile: There are two syntaxes. This is the traditional 1. Whenever you see code inside square brackets, you are sending a message to an object or a class. Dot Syntax The dot syntax for getters and setters is new in Objective-C 2. The dot syntax should only be used setters and getters, not for general purpose methods. The first is the one you saw before: In many cases, though, you need to create an object using the manual style: The first is the alloc method called on NSString itself. This is a relatively low-level call which reserves memory and instantiates an object. The second piece is a call to init on the new object. The init implementation usually does basic setup, such as creating instance variables. The details of that are unknown to you as a client of the class. In some cases, you may use a different version of init which takes input: However, you may not always be working with an environment that supports garbage collection. In that case, you need to know a few basic concepts. If you create an object using the manual alloc style, you need to release the object later. You should not manually release an autoreleased object because your application will crash if you do. Here are two examples: It typically comes in two parts. The class interface is usually stored in the ClassName. The implementation is in the ClassName. The class is called Photo, so the file is named Photo. The import directive automatically guards against including a single file multiple times. The interface says that this is a declaration of the class Photo. The colon specifies the superclass, which is NSObject. Inside the curly brackets, there are two instance variables: Both are NSStrings, but they could be any object type, including id. Finally, the end symbol ends the class declaration. By default, the compiler assumes a method returns an id object, and that all input values are id. All methods must appear between these two statements. The first is a reference to the existing object, and the second is the new input object. In a garbage collected environment, we could just set the new value directly: There are actually two ways to free a reference to an object: The standard release will remove the reference immediately. The autorelease method will release it sometime soon, but it will definitely stay around until the end of the current function unless you add custom code to specifically change this. The autorelease method is safer inside a setter because the variables for the new and old values could point to the same object. This may seem confusing right now, but it will make more sense as you progress. We can create an init method to set inital values for our instance variables: This is a single equals sign, which assigns the result of [super init] to self. This essentially just asks the superclass to do its own initialization. The if statement is verifying that the initialization was successful before trying to set default values. Dealloc The dealloc method is called on an object when it is being removed from memory. This is usually the best time to release references to all of your child instance variables: The last line is very important. We have to send the message [super dealloc] to ask the superclass to do its cleanup. The dealloc method is not called on objects if garbage collection is enabled. Instead, you implement the finalize method. All you have to do is keep track of your references, and the runtime does the actual freeing of memory. So if you used alloc once and then retain once, you need to release twice. But in practice, there are usually only two reasons to create an object: To keep it as an instance variable 2. To use temporarily for single use inside a function In most cases, the setter for an instance variable should just autorelease the old object, and retain the new one. You then just make sure to release it in dealloc as well. So the only real work is managing local references inside a function. If you create an object any other way, do nothing. We only need to release the object created with alloc: NSLog "The current date and time is: You can override the description method in your class to return a custom string. Properties are a feature in Objective-C that allow us to automatically

generate accessors, and also have some other side benefits. The "retain" in the parenthesis specifies that the setter should retain the input value, and the rest of the line simply specifies the type and the name of the property. The compiler will fill in whichever method is missing. There are many other options for the property declarations, but those are outside of the scope of this tutorial. The difference is that you can call methods on nil without crashing or throwing an exception. If you call a method on nil that returns an object, you will get nil as a return value. We can also use this to improve our dealloc method slightly: If we just directly set the value like this, there would be a memory leak: This is particularly useful because you can add methods to built-in objects. If you want to add a method to all instances of NSString in your application, you just add a category. The name can be whatever you want, though it should communicate what the methods inside do. Keep in mind this is not a good implementation of URL detection. The following code will print "string1 is a URL" in the console: You can, however, use categories to override existing methods in classes, but you should do so very carefully. Remember, when you make changes to a class using a category, it affects all instances of that class throughout the application. Wrap Up This is a basic overview of Objective-C.

Chapter 5 : Cocoa Dev Central: Learn Objective-C

If you want to learn Objective-C, Kochan's book is the way to go. Hillegass does talk about Objective-C, but it's only one chapter and assumes a good basis of prior object-oriented programming experience (if you already know C++ and/or Java, it's probably enough).

Next About Objective-C Important: This document is no longer being updated. For the latest information about Apple SDKs, visit the documentation website. Objective-C inherits the syntax, primitive types, and flow control statements of C and adds syntax for defining classes and methods. It also adds language-level support for object graph management and object literals while providing dynamic typing and binding, deferring many responsibilities until runtime. At a Glance This document introduces the Objective-C language and offers extensive examples of its use. Although the framework classes are separate from the language, their use is tightly wound into coding with Objective-C and many language-level features rely on behavior offered by these classes. This interface includes the public properties to encapsulate relevant data, along with a list of methods. Method declarations indicate the messages that an object can receive, and include information about the parameters required whenever the method is called. If you do have the original source code for a class, you can use a class extension to add new properties, or modify the attributes of existing properties. Class extensions are commonly used to hide private behavior for use either within a single source code file, or within the private implementation of a custom framework. Customizing Existing Classes Protocols Define Messaging Contracts The majority of work in an Objective-C app occurs as a result of objects sending messages to each other. Often, these messages are defined by the methods declared explicitly in a class interface. Objective-C uses protocols to define a group of related methods, such as the methods an object might call on its delegate , which are either optional or required. Any class can indicate that it adopts a protocol, which means that it must also provide implementations for all of the required methods in the protocol. The NSString class is used for strings of characters, the NSNumber class for different types of numbers such as integer or floating point, and the NSValue class for other values such as C structures. You can also use any of the primitive types defined by the C language, such as int, float or char. Blocks are often used to simplify common tasks such as collection enumeration, sorting and testing. They also make it easy to schedule tasks for concurrent or asynchronous execution using technologies like Grand Central Dispatch GCD. Working with Blocks Error Objects Are Used for Runtime Problems Although Objective-C includes syntax for exception handling, Cocoa and Cocoa Touch use exceptions only for programming errors such as out of bounds array access , which should be fixed before an app is shipped. All other errorsâ€”including runtime problems such as running out of disk space or not being able to access a web serviceâ€”are represented by instances of the NSError class. Your app should plan for errors and decide how best to handle them in order to present the best possible user experience when something goes wrong. Method names, for example, start with a lowercase letter and use camel case for multiple words; for example, doSomething or doSomethingElse. In addition, there are a few conventions that are required if you wish to take advantage of language or framework features. Additionally, you should become familiar with Xcode before trying to follow the exercises at the end of most chapters in this document. If you have knowledge of another higher-level programming language, such as Ruby or Python, you should be able to follow the content. Reasonable coverage is given to general object-oriented programming principles, particularly as they apply in the context of Objective-C, but it is assumed that you have at least a minimal familiarity with basic object-oriented concepts. See Also The content in this document applies to Xcode 4. For more information about Xcode, see Xcode Overview. Objective-C apps use reference counting to determine the lifetime of objects. In addition to the compiler, the Objective-C language uses a runtime system to enable its dynamic and object-oriented features. Terms of Use Privacy Policy Updated: Please try submitting your feedback later. Thank you for providing feedback! Your input helps improve our developer documentation. How helpful is this document?

Chapter 6 : 4 Free E-Books on Learning Objective-C, the Programming Language of iOS and OSX - ReadW

The Objective-C language is a superset of ANSI C with special syntax and runtime extensions that make object-oriented programming possible. Objective-C syntax is uncomplicated, but powerful in its simplicity.

Quartz is a vector drawing API, which means that the drawing code is independent of the pixels on the screen. This is in contrast to usual image files, which has a finite number of pixels. The end result is that graphics drawn with Quartz will look especially sharp on Retina Displays, without resorting to additional resource files. Quartz 2-D provides low- level, lightweight 2-D rendering with unmatched output fidelity regardless of display or printing device. Custom drawing Quartz is used to do a lot of custom drawing across iPhone apps and can distinguish apps and their custom interfaces. For example, the Stocks app uses to draw the graphs the reflect the stock price over a specific interview. It is not possible to ship the app with every possible graph already drawn out as a regular PNG image; the app has to create the graphics itself. It does this using the Quartz APIs. All the code you write for Quartz will consist of C functions. How Quartz Works Two fundamental patterns govern most Quartz functions. Quartz, like many other rendering engines, is state-based. Generally, you set some values, such as color or line- width, and those same values continue to be used until you set other values. This simply means that content drawn first will appear underneath newer content; in other words, new content can obscure older content. Therefore, the order in which you draw things matters. Most Quartz functions are relative to a specific context. A context is simply a virtual canvas which you can draw to. This could be the screen, a PDF, a bitmap image file, or a custom context. Most Quartz functions take a context as an argument, to know which context to draw to. In fact, this is how printing works: This was a technological breakthrough when it was inventedâ€" to be able to use the same code to draw on-screen as to the printer. There are three general types of functions in Quartz. You begin by drawing to the context. This creates an abstract, transparent representation in the context. At this point, there is a data structure in memory, but nothing is actually shown. This allows you to use Quartz as an underlying structure for more complex drawing, such as displaying text along a curve. Quartz functionality should be immediately familiar to graphic designers. Nearly anything you can do with vector drawing programs, such as Adobe Illustrator, you can do with Quartz. Drawing in a view begins in the drawRect: You put Quartz calls in drawRect: However, you should never call drawRect: If you want to force the system to redraw an area of the screen, you can call setNeedsDisplay on the view, and it will be redrawn according to the code in drawRect:. Drawing Basics Most drawing in Quartz begins with a graphics context. You should save this context in a local variable for all subsequent drawing calls: You should also pop it when done. In addition, the system automatically sets up the context each time drawRect: No guarantee is made about the state of the context across multiple method calls; therefore, you should not save the context in a local variable. Quartz, as you might imagine, is primarily based on x-y points. All points are positive unless you do some custom mapping for whatever reason. On the iPhone, the point 0,0 is the top-left point of the view, which follows the convention of web design and makes sense in many cases. Quartz on the desktop uses the reverseâ€"like in geometry textbooks, 0,0 is located at the bottom-left point. This means that, without any modification of points, Quartz code from the iPhone will appear upside-down on the desktop, and the opposite is true as well. All drawing in Quartz begins with rectangles. They are represented by the CGRect structure, containing an origin point and a size; the former is comprised of an x- and y-value float, while the latter is composed of a width and height. You can then stroke or fill the rectangles directly. To draw other shapes, you can either define a free-form path, or start with a bounding rectangle. You do this by defining paths. For example, to draw a triangle you could use something like this:

Chapter 7 : Learn Objective-C: A Quartz Primer

Learning Objective-C: A Primer (ä¸æ–‡ç‰ˆ) Objective-C è¯è¨€æ˜¯ä¸€é—¨ç®€å••çš„è®¡ç®—æœºè¯è¨€ï¼Œå®ƒè¾è®¡çš„ç›®æ ‡æ˜¯ä¸®åŠ©å¼€å•'è€…ç²¾å·§çš†Ÿç»ƒåœ°è¿›

è¡Œé•¢å•'å¯¹è±¡.

Chapter 8 : Learning Objective-C - Free download, Code examples, Book reviews, Online preview, PDF

Objective-C is built on top of the C programming language. You will need to have an understanding of parts of C before you can get to the particulars of Objective-C.

Chapter 9 : About Objective-C

Learn Objective-C on the Mac: For OS X and iOS, Second Edition updates a best selling book and is an extensive, newly updated guide to Objective-C. Objective-C is a powerful, object-oriented extension of C, making this update the perfect follow-up to Dave Mark's bestselling Learn C on the Mac.