

## Announcements

- **Lab 3 is due tonight by 11:59 PM**
- **Lab 4 is due on Monday October 23<sup>rd</sup>**

## Today's Topics

- **Recap of SQLite**
- **XML and JSON**
- **Web Services**
- **WKWebKit**
- **Threading**

# SQLite

# SQLite

- **Complete SQL database in an ordinary file**
- **Simple, compact, fast, reliable**
- **No server**
- **Great for embedded devices**
  - Included on the iPhone platform

## When Not to Use SQLite

- **Multi-gigabyte databases**
- **High concurrency (multiple writers)**
- **Client-server applications**
- **“Appropriate Uses for SQLite”**  
<http://www.sqlite.org/whentouse.html>

## More on SQLite

- **“SQLite in 5 Minutes Or Less”**  
– <http://www.sqlite.org/quickstart.html>
- **“Intro to the SQLite C Interface”**  
– <http://www.sqlite.org/cintro.html>
- **FMDB SQLite Wrapper**  
– <https://github.com/ccgus/fmdb>

## Core Data

- **Object-graph management and persistence framework**
  - Makes it easy to save and load model objects
    - Properties
    - Relationships
  - Higher-level abstraction than SQLite or property lists

## Web Services

## Your Application & The Cloud

- **Store & access remote data**
- **May be under your control or someone else's**
- **Many Web 2.0 apps/sites provide developer API**

## Integrating with Web Services

- **Non-goal of this class: teach you all about web services**
  - Plenty of tutorials accessible, search on Google
- **Many are exposed with XML or JSON**
- **High level overview of parsing these types of data**

# XML

## Options for Parsing XML

- **XMLParser**
  - Event-driven API
  - <https://developer.apple.com/reference/foundation/xmlparser>
- **SWXMLHash**
  - Simple Swift XML Parsing
    - <https://github.com/drmohundro/SWXMLHash>

# JSON

## JavaScript Object Notation

- **More lightweight than XML**
- **Looks a lot like a property list**
  - Arrays, dictionaries, strings, numbers
- **Open source json-framework wrappers for Swift and Objective-C**

## What does a JSON string look like?

```
{  
  "instructor" : "Todd Sproull",  
  "students" : 20,  
  "itunes-u" : true,  
  "midterm-exam" : null,  
  "assignments" : [ "WhatATool",  
                    "HelloPoly"]  
}
```

## SwiftJSON Demo



## More on JSON

- **“Introducing JSON”**
  - <http://www.json.org/>
- **SwiftyJSON**
  - <https://github.com/SwiftyJSON/SwiftyJSON>
- **JSON Editor**
  - <http://www.jsoneditoronline.org/>

## Recap

- **Property lists**
  - Quick & easy, but limited
- **Archived objects**
  - More flexible, but require writing a lot of code
- **SQLite and Core Data**
  - Elegant solution for many types of problems
- **XML and JSON**
  - Low-overhead options for talking to “the cloud”

# Firestore Demo

# Web Content in iOS

## Displaying Web Content

- **Web content can be displayed with WKWebView**
  - Introduced in iOS 8, part of WKWebKit Framework
    - Replaces `UIWebView`
- **Content can be**
  - local HTML string
  - local raw data + MIME type
  - remote URL
- **Leverages WebKit**
  - full WK functionality not currently exposed
  - simple API for loading & navigating
  - delegate for some control
  - Same JavaScript engine that powers Safari

## WKWebView

- **WKWebView subclass, configure in Storyboard or in code**
    - Feed it data to display
- ```
func loadHTMLString(_ string: String, baseURL:URL?) -> WKNavigation?
```
- ```
func load(_ data: Data,  
mimeType: MIMEType: String,  
characterEncodingName: String,  
baseURL: URL )-> WKNavigation?
```
- **Or give it a URL request**

```
func load(_ request: URLRequest) -> WKNavigation
```
  - **WKNavigation**
    - Object that contains information for tracking the loading progress of a webpage
  - **What's this URLRequest?**
    - Encapsulates a URL to load and caching policy for fetched data
    - Older versions of iOS used an `NSURL` and `NSURLRequest`

## WKWebView

- Properties and actions you'd expect from a web view

`isLoading: Bool`  
`canGoBack: Bool`  
`canGoForward: Bool`  
`reload()`  
`stopLoading()`  
`goBack()`  
`goForward()`

- A couple others that are handy  
`estimatedProgress: Double`  
`evaluateJavaScript(_:completionHandler:)`

## WKNavigationDelegate

- Callbacks for load progress  
`webView(_: didCommit: )` //called when content starts arriving  
`webView(_: didFinish: )` //called when navigation is complete
- Error handling  
`webView(_: didFail: withError: )`
- Navigation Loading Policy  
//Decides whether to allow or cancel a navigation  
`webView(_: decidePolicyFor: decisionHandler:)`

## Demo WKWebView

## Multithreading in iOS

- Work done with Queues
- Functions (closures) are assigned as units of work to the queues
- Queues execute on a CPU thread
- Queues are either serial or concurrent
- Queues are synchronous or asynchronous

## Types of Queues

- **Main Queue**
  - Special serial queue where all UI-Activity happens
  - Non-UI actions should take place on background queue
    - Important to do this to free up main queue
- **Global Queues**
  - Four queues shared by the system with different priority levels
- **Custom Queues**
  - User generated queues with custom attributes (name, priority level, etc)

## Multithreading (from CS193P)

- **Executing a function on another queue**

```
let someQueue = DispatchQueue(label: "name")

someQueue.async { /* do work here */ }
```
- **The main queue (serial queue)**
  - DispatchQueue.main
- **All UI work done on main queue**
- **All time intensive code or synchronous (blocking) done on another queue**
- **Swift 3 provides global queues with different priorities**

```
DispatchQueue.global(qos: .userInitiated).async {

    //do non-UI stuff that may take time
    DispatchQueue.main.async {
        // Call UI functions with with results from other queue
    }
}
```

## Multithreading (from CS193P)

- **Specifying QOS for queues**

- `userInteractive` //quick and high priority
- `userInitiated` //high priority, may take some time
- `utility` //long running
- `background` //user not concerned (prefetching)

```
let queue1 = DispatchQueue(label: "low priority" qos:  
    "DispatchQueue.background")
```

```
let queue2 = DispatchQueue(label: "high priority" qos:  
    "DispatchQueue.userInteractive")
```

## QoS Demo

## Multithreading (CS 193P)

- **Multithreaded iOS API**

- Many iOS APIs execute on a queue other than the main queue
- These APIs typically provide a closure as an argument, which is called upon completion of the method
- If you want to update the UI, you will need to dispatch back to the main queue

```
DispatchQueue.global.async {  
    DispatchQueue.main.async {  
        // Call UI functions with results from other queue  
    }  
}
```

## Multithreading DEMO



## More on Concurrent Programming

- **Grand Central Dispatch (GCD)**

- [https://developer.apple.com/library/mac/documentation/Performance/Reference/GCD\\_libdispatch\\_Ref/index.html](https://developer.apple.com/library/mac/documentation/Performance/Reference/GCD_libdispatch_Ref/index.html)
- <https://developer.apple.com/videos/play/wwdc2016/720/>

- **GCD Tutorial with Examples**

- <https://www.raywenderlich.com/148513/grand-central-dispatch-tutorial-swift-3-part-1>