iOS Dev Accelerator Week3 Day2

- Custom App URL Schemes
- OAuth
- KeyChain
- Generics
- Ternary Operator
- App Delegate/Root ViewController
- Big O

Homework Review

Custom URL Scheme

Registering a custom URL scheme

- A custom URL scheme for your app allows your app to be opened by other apps.
- This includes safari, which is how websites offer to open pages in apps, if you have it installed.

Registering a custom URL scheme

- Go to your info.plist and choose Add Row
- Select URL types from the drop down (its an array with one dictionary)
- In the item 0 dictionary, add an entry for the key ‘URL Identifier’
- This entry will be the name of the custom url scheme you are defining. It is recommended to ensure uniqueness that you use reverse DNS ‘com.yourCompany.yourApp’
- Add another entry from the drop down called ‘URL Schemes’ which is an array
- For Item 0, give it the string you want to be your URL Scheme. For example, if you type MyApp. The custom URL for your app will be MyApp://

Try it out!

- Launch your app after editing your plist with your URL scheme.
- Now launch Safari from your simulator, and enter in your app's URL into the URL bar.
- Your app should now open.

Parsing information that is passed with URL's

- Often times, and especially with OAuth, your app's url will be called with extra parameters. Typically this is a token or flag.
- There is a method you can implement in your app delegate to intercept these URL calls:

```
- (BOOL)application:(UIApplication *)application
  openURL:(NSURL *)url
  sourceApplication:(NSString *)sourceApplication
  annotation:(id)annotation
```

- We will pass the URL they passed back to us to our network controller for parsing

OAuth

OAuth

- OAuth is is an authentication protocol that allows two apps to interact and share resources.
- There are 3 actors in the OAuth workflow:
  - The service provider: Ex: GitHub
  - The consumer: Ex: Our app
  - The user
- The main benefit is that the user never shares his account and password with the consumer app.

OAUTH Workflow

- Step 1: The user shows intent by attempting an action from the consumer app to the service provider (aka GitHub)
- Step 2: Our app (The consumer) redirects the user to the service provider for authentication
- Step 3:The user gives permission to the service provider for all the actions the consumer should be able to do on his/her behalf (ex: posting to their timeline, accessing their twitter photos, etc)
- Step 4:The service provider returns the user to the consumer app, with a request token
- Step 5:The consumer now sends the request token, together with its secret key to the service provider in exchange for an authentication token
- Step 6 (repeating): The user performs actions and passes the authentication token with each call to prove who he is

Callback URL

- The callback URL is just the callback entry point of your app.
- The service provider performs an HTTP redirect to get the user back to the consumer app.
- In addition to the URL of your app, the callback url will have the authorization code appended to it. It is up to your app to parse this out and use it in completing the OAuth workflow.
- All apps can be launched from either another app or from the browser itself.

Demo

Keychain

- Now that we are getting back this authentication token, we need some way to persist it to disk, since it wont change from session to session, and we don't want to make our user go through that process each time they launch the app
- You may have thought, “well user defaults would be great for this!”….WRONG
- NSUserDefaults is saved in plain text to disk as a property list, so saving passwords, tokens, and other sensitive information is bad. Instead, save it to the Keychain.
- Unfortunately, the Keychain API is a bit difficult to use, especially with Swift

Keychain API

- First, any class that works with the KeyChain API must import the Security framework
- You then simply have 4 functions to use based on what you need to do:
  - SecItemAdd: use to add an item to Keychain
  - SecItemUpdate: use to modify an existing item in the Keychain
  - SecItemCopyMatching: use to query Keychain for an item
  - SecItemDelete: use to delete an item from the Keychain

Keychain API

- Sounds pretty simple, right!? WRONG AGAIN
- Keychain is a implemented as a C based API, so it uses some types that a lot of Cocoa Devs have never needed to use before, so it can be very daunting to use
- Plus, we are going to use it in Swift, which adds more complexity since Swift has some weird looking syntax when dealing with C

Keychain Queries

- Anytime you access the Keychain via one of the Sec functions listed a few slides back, you need to pass in a Keychain Query
- A Keychain Query is just a dictionary, but it requires certain keys to be present in order for it to work
- The following keys are important:
  - kSecReturnData : pass in kCFBooleanTrue when you are trying to retrieve the item from Keychain, otherwise you can omit this key
  - kSecClassKey: key whose value is the item's class code. Class codes include generic password, **internet password**, certificate, key, identify.
  - kSecAttr*: This is where you specify information about the item you are looking for or are storing. If you specify your class as a generic or internet password, you will want to use kSecAttrAccount
  - kSecValueData: For this key, you pass in the actual data you are trying to store, should be a string.

Demo

Unmanaged<T>

- First you will see the type Unmanaged used. Unmanaged is a struct Swift uses when ever you use C or Objective-C API's that have not been annotated for Swift.
- Swift wraps any objects coming from these APIs in these unmanaged structs, telling you that Swift and the Compiler are NOT going to automatically manage the memory for this object
- Whenever you receive an unmanaged object from an API, you should immediately convert it to a memory managed object before you work with it.
- How do you do that? It depends!

Retained vs Unretained

- To get a memory managed version of the object you want to work with, you can call:
  - takeRetainedValue() : call this if the object you are working with is already retained
  - takeUnretainedValue() : call this if the object you are working with has not been retained.
- How do you know if the object you got from a function is retained or not?
- Via naming conventions! If you see the word Copy or Create in the name of the function, you can assume the object coming back has been retained for you.
- And if you get back an unmanaged, retained object, you are in charge of releasing it!

Demo

AppDelegate & RootViewController

Swift Generics

Generics

- Generics are a feature in Swift that allow you to write flexible, reusable code that you can constraint to specific requirements.
- Generics helps avoid duplicate code
- We have been using generics this entire time, you just might not have realized it.
- Arrays and Dictionaries are both collections implemented with generics
- **Generics can be used to define generic functions and/or generic types**

A generic function

```
func swapTwoValues<T>(inout a: T, inout b: T) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

A generic type

```
struct Stack<T> {
    var items = [T]()
    mutating func push(item: T) {
        items.append(item)
    }
```

<T>

- When you see <T>, that is a type parameter.
- Type parameters specify and name a placeholder type, and written immediately after a generic function's name, or a generic type's name
- Once you have this declared, you can use this type as a paremeter, return type, a regular type annotation or property type.
- In all casses, the placeholder type represented by the type parameter is replaced with an actual type whenever the function or type is called/used.
- If you only have one type parameter, traditionally T is used, but you can name it anything.

Demo

Type Constraints

- Sometimes you may want to use generics, but constrain the type parameter so that only certain types can be used.
- You can use type constraints to specify a type parameter must inherit from a certain class, or conform to a particular protocol
- Swift's Dictionary struct is a great example of this. It places a type constraint on the type parameter for keys. It specifies that any type used for a key must be hashable (more on hashing later in the course).

Type Constraints

```
func someFunction<T: SomeClass, U: SomeProtocol>
        (someT: T, someU: U) {
    // function body goes here
}
```

In this example, whatever type passed in for T must inherit from SomeClass, and whatever type passed in for U must conform to SomeProtocol

Demo

Ternary Operator

Ternary Operators

- Ternary Operators in Swift allow you to do if-else conditionals in a short hand syntax
- Ternary Operators take this basic form:
  - expression_to_evaluation ? expression_if_true : expression_if_false

Without Ternary

```
let x = 10
let y = 9

if x == y {
  println("they match!")
} else {
  println("no match")
}
```

With Ternary

```
let x = 10
let y = 9

x == y ? println("they match!") : println("no match")
```

Ternary Operators

- Its common to see ternary operators used to set variables or constants:

```
let result = x < 11 ? "less than" : "greater than"
```

- You can nest ternary operators:

```
let result = i % 2 == 0 ? "a" : i % 3 == 0 ? "b" : i % 5 == 0 ? "c" :
  i % 7 == 0 ? "d" : "e"
```

  - Don’t do this. One ternary operator can already be hard to read
  - Why should you use ternary? It makes our code more concise
  - If your conditions have more than one line, don’t use the ternary operator

Demo

App Delegate

Application Delegate Protocol

- The AppDelegate object in your app is simply a regular object that conforms to the UIApplicationDelegateProtocol
- This protocol defines methods that are called by the singleton UIApplication object in response to important event in the lifetime of your app (going from foreground to background, receiving a local notification, etc)
- When your app launches, UIKit automatically creates an instance of the app delegate class provided by Xcode when you initially created your project.
- The App delegate is effectively the root object of your app.

Application Delegate Protocol

- Crucial Roles:
  - It contains your apps startup code
  - It responds to changes in your app’s states (foreground to background, etc)
  - It responds to notifications originating from outside your app, like remote notifications, low memory warnings, download completions, etc
  - You can store your app’s central data objects or any content that does not have an owning view controller

Application did finish launching with options

- This method is called after your app has finished its initialization process, but before the user sees any view controllers.
- This is a great spot to change the root view controller of the app based on some preferences or saved data
- For our app, if you already have the oauth token saved to user defaults, go straight to the main menu view controller. If not, go to the login view controller.

UIWindow

- UIWindow is a class that manages and coordinates the views an app displays on the device screen
- UIWindow is a subclass of UIView
- Windows have 2 main jobs, provide area for displaying other views, and distribute events to the views
- Windows have Z axis, which dictates which windows overlap other windows. For example an alert view is the highest z window.
- Only window at a time is allowed to receive keyboard and touch related events. This window is called the keyWindow.

Working with UIWindow

- Most apps don't need any code directly manipulating window objects.
- If you do, its most commonly because you simply want to change the root view controller of your app.
- UIWindow has a rootViewController property, which you can set to change the root view controller.
- To access the window that is displaying the scenes of your app, you access the window property of the app delegate

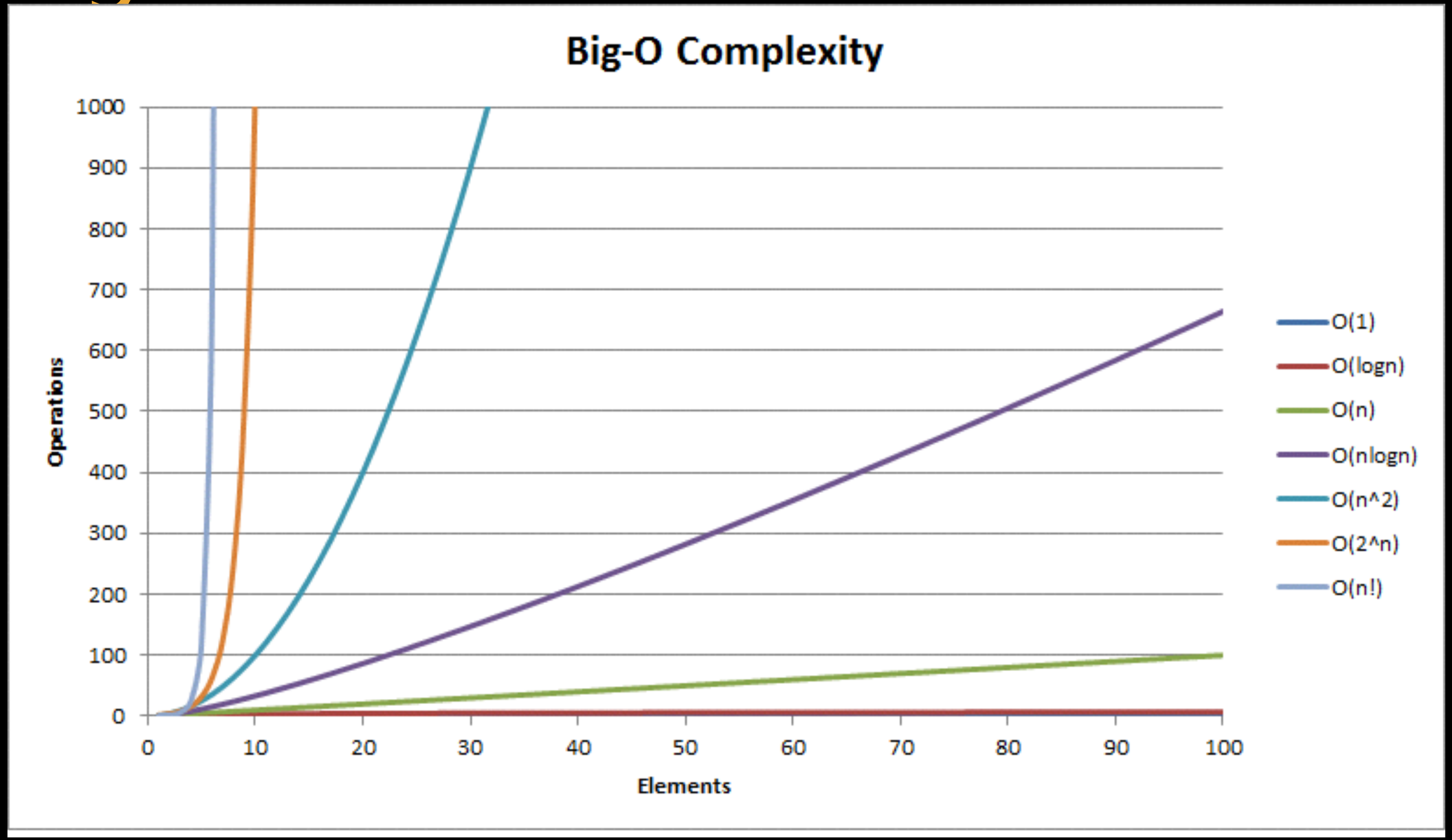
Demo

Big O

Big O notation

- In computer science, big o notation is used to measure the efficiency of algorithms.
- The O stands for order, because because the growth rate of an algorithm is also refereed to as the order of the algorithm.
- In most big o notations, you will see the variable N. N refers to the size of the data set on which the algorithm is being performed. So if we are searching through an array of 10 strings, N == 10.
- Big-O usually refers to the worst case scenario.

Big O notation



O(1) – Constant Time

- The running time of any ‘simple’ operation is considered constant time, or O(1). Things like setting properties, checking bools, simple math equations are constant time.
- A constant time efficiency is awesome!

O(N) – Linear

- An example of a O(N) algorithm is a for loop.
- The loop executes N times. Lets say you are looping through an array of 10 Ints looking for a specific value. So N is 10 here. At worst case, the value you are looking for is at the end of the array, and it took you N (or 10) operations to find it.
- The operation inside of the for loop is a constant time value check. So you might think the Big-O notation of this algorithm is O(N) * O(1). But when you are working with Big-O, you can drop constant times since they are trivial, so the Big-O of a regular for loop is just O(N)

O(N^2)

- O(N^2) refers to any algorithm whose Big-O is the square of the input data set.
- An example of this would be a nested for loop.
- Lets say we are searching for duplicates in an array of Strings. For each string in the array, we search through every other string and check if the values are the same. So if we have 7 Strings, this operation will run 49 times, or 7x7, or 7^2.

O(log N)

- A good rule of thumb: if your algorithm cuts the data set in half for each step of the algo, you are probably working in O(log N).
- Looking at our graph, we can see algorithms with this notation peaks at the beginning and then quickly levels out as the data size increases.
- O(log N) is great! Even when you double your N, the worst case time to run the algorithm only increases by a small amount. The classic example of this is a Binary Search Tree.