iOS Dev Accelerator
Week 5 Day 2

- Blocks
- Objective-C Constants
- CoreLocation
- Map Annotations
- Parse GeoPoints

Blocks

- Introduced in ObjC 2.0
- Serve the exact same purpose as Closures in Swift
- Used extensively in Cocoa APIs that accept completion blocks
- Can be anonymous or stored local variables or properties, same as closures
- Very easy to introduce retain cycle, if you're not careful. (same with closures, surprise!!)
- Syntax not easy to remember:

http://goshdarnblocksyntax.com

Where does the weird block syntax come from?

- Objective-C uses the exact same syntax for declaring pointers as C does:

```
int *i;
NSString *name;
```

- You can declare a pointer to a function:

```
float (*myFunction)(Int);
```

- And heres how a block of that same type (a function that takes in an Int and returns a float) is declared:

```
float (^myFunction)(Int);
```

*sourced from arigrant.com

Defining the block

- So we know how to declare a block:

```
float (^myFunction)(Int);
```

- But what about when we need to define a block's functionality?
- Heres the syntax:

```
^float (int *i) {

return i * 2.0;

}
```

Blocks

- Block are denoted with the caret operator ^
- Declare a return type; typically void on completion blocks
- Include types of parameters. Do not need to be named in method prototype

```
- (void)loadRoutes:(void (^)(NSArray *, NSError *))success;
```

```
[[RFRoutesModel sharedModel] loadRoutes:^(NSArray *routes, NSDictionary *err) {
    //
}];
```

Demo

__block

- When you capture a non-object variable (structs, primitives), only the value is captured.
- If you need to be able to change the value of a captured variable from within a block, you can use the __block storage type modifier
- This makes it so the variable lives in storage that is shared between the lexical scope of the original variable and any blocks declared within that scope.

Demo

Weak self and blocks

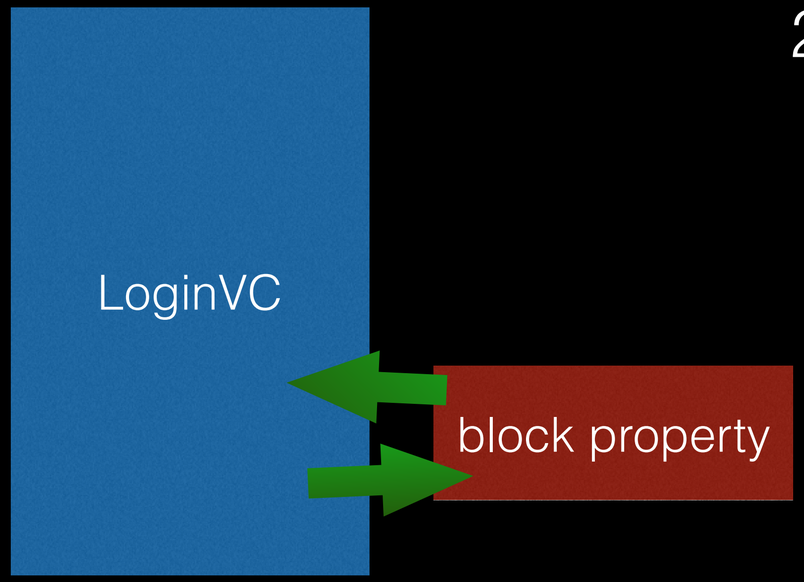
- When referring to self inside a block (or closure), best practice is to create a weak reference of self before the block and use that reference instead of self.
- Since blocks and closures capture strong references to all objects accessed in their bodies, if the object represented by self were to get a strong reference to the block or closure, then we would have a retain cycle.
- We can use the __weak modifier in Objective-C to avoid this

Weak self and blocks

1) LoginVC declares a strong (or copy) property of a block:

```
@interface ViewController ()

@property (copy,nonatomic) float(^ myBlock)(int);
```



2) self is referenced in the declaration of the block, the block now owns self (which is the LoginVC)

```
- (void)viewDidLoad {
  [super viewDidLoad];

  self.myBlock = ^float (int * I) {
    self.title = @"Login";
    return 31;
  };
```

3)RETAIN CYCLE!!!!

Demo

Objective-C Constants

Using const

- Declare constants using the const keyword
- Any type can be a constant
- Value cannot change, once declared.
- Think of let keyword in Swift

```
NSString * const ReuseIdentifier = @“ReuseIdentifier";

int const limit = 100
```

Sharing const

- Expose constants in a header file to use from multiple points
- Define the actual value of the constants in an implementation file
- Useful technique to define constants and use across the app

Where to put constants you may need in multiple places

- Create Constants.h and Constants.m
- In Constants.h, expose the const variables

```
extern NSString *const MyFirstConstant;
```

The extern keywords make this a globally available constant as long as you import constant.h

- Extern just lets the compiler know, and anyone reading your code that this variable is declared here, but defined somewhere else.
- In Constants.m, define the const variables value

```
NSString *const MyFirstConstant = @"FirstConstant";
```

Where to put constants you only need in one place

- Put it in the .m
- Before the implementation block of the class

CoreLocation

Core Location

- Core Location provides several services that you can use to get and monitor the devices current location:
  - The significant-change location service provides a low power way to get the current location and be notified when significant changes occur .
  - The standard location service offers a highly configurable way to get current location and track changes.
  - Region monitoring lets you monitor the boundary crossing defined by geographical regions and bluetooth low energy beacon regions (iBeacons!).

Location services availability

- Situations where location services might not be available
  - The user disables location services in the settings app or system preferences
  - the user denies location services for a specific app
  - the device is in airplane mode and unable to power up the necessary hardware
- You should always call locationServicesEnabled class method on CLLocationManager before attempting to start services. If it returns NO and you attempt to start the services anyway, the user will be prompted to turn on the services, which may be annoying after one time.

CoreLocation Authorization

- CLLocationManager has a class method for retrieving the authorization status: authorizationStatus()
- Also has a delegate method when the authorization status changes: locationManager:didChangeAuthorizationStatus:
- You can request authorization for always running (foreground+background) or just foreground.
- Upon requesting, the user will be presented with the text you have stored in your NSLocationAlwaysUsageDescription or NSLocationWhenInUseUsageDescription keys in your info.plist. **You must have these keys for the system to ask the user for permission.**

Standard Location Service

- Most common way to get a user's current location since its available on all devices.
  - Before activating, you specify the desired accuracy and the distance that must be traveled before reporting a new location.
  - Once you start the service, it uses those parameters to determine which hardware to use.
  - Create an instance of CLLocationManager class to get your services setup.
  - You will need authorization before activating it.

CLLocationManager

- Configure the desiredAccuracy and distanceFilter properties before starting the services.
- The distance filter is used for movement. When the location manager has detected movement that is larger than the distance filter, an update is delivered. Not setting it means you want updates for any small movement, which will be a lot.
- Desired accuracy tells the location manager how accurate readings should be that are delivered to the delegate. The more accurate you want, the more battery power your app will eat up. Use the absolute minimum your app can get away with.
- To start the services, make sure you have a delegate set and then call startUpdatingLocation method.
- The delegate will now be updated as location data becomes available.
- Call stopUpdatingLocation to stop the updates.

Significant-Change Location Service

- Provides accuracy that is good enough for most apps, and saves significantly more power than regular location services.
- This service uses Wi-Fi to determine the user's location and report changes in that location.
- To begin monitoring significant changes, setup your CLLocationManager with a delegate, and then call startMonitoringSignificantLocationChanges method.

Demo

Receiving Location Data

- The way you receive location events is the same whether you use the standard or the significant-change location services.
- The location manager reports to the delegate by sending locationManager:didUpdateLocations: method.
- If theres an error with the location, it sends locationManager:didFailWithError

CLLocation

- The delegate method sends an array of CLLocation objects.
- CLLocation represents location data.
- Incorporates geographical coordinates and altitude of the devices location along with accuracy measurements and when the measures were taken.
- In iOS, this object also has data about the speed and heading of the device.

CLLocation Properties

- coordinate : CLLocationCoordinate2D - struct with two values: latitude and longitude, both in degrees. Positive values north of equator and negative south of the equator.
- altitude : CLLocationDistance - A double, positive values indicate above the sea level, negative below.
- course : CLLocationDirection - The direction in which the device is traveling. Measured in degrees starting at due north and continuing clockwise. North is 0 degrees, east is 90,etc. Negative means invalid.
- horizontalAccuracy : CLLocationAccuracy - A double, the lat and long identify the center of the circle, this value indicates the radius of that circle.
- timeStamp : NSDate - can be used to figure out how ‘stale’ the location update is. Use timeIntervalSinceNow.

CLLocationDistance

- CLLocation has an instance method that takes in another CLLocation and returns a CLLocationDistance.
- CLLocationDistance is a double and is in meters.

Demo

Map View Annotations

MKMapView Annotation

- “Annotations display content that can be defined by a single coordinate point”
- User’s current location, a specific address, single points of interest, etc.
- Remain fixed to the map.
- “Map Kit separates the data associated with an annotation from its visual presentation on the map” (allows for great performance, even with hundreds of annotations)

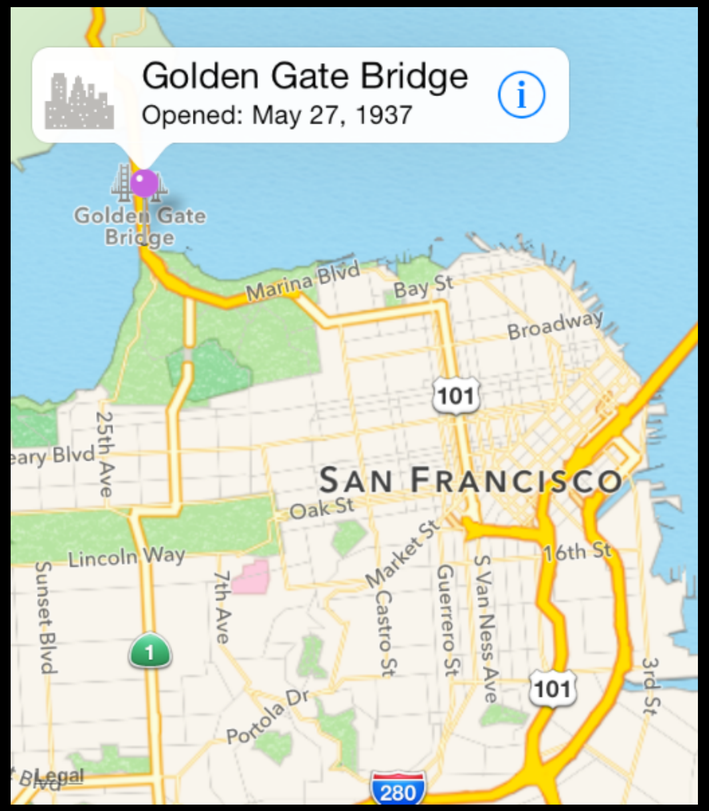
Adding Annotations

- To display an annotation, you need two objects:
- An annotation object, which is an object that conforms to the MKAnnotation protocol and manages the data for the annotation. Typically pretty small object.
- An annotation view, which is a view used to draw the visual representation of the annotation on the map.

Adding Annotations

1. Instantiate an MKPointAnnotation instance, or you own custom class that conforms to MKAnnotation protocol.
2. Give that annotation instance data (coordinates, optional title and subtitle for your custom class)
3. Call addAnnotation: or addAnnotations: on your map view.
4. **Optional:** Implement mapView:ViewForAnnotation delegate method on your map view's delegate to return an AnnotationView for a given annotation. If you don't implement this method, you will get default behavior of a plain red pin with no custom behavior.

Annotation Callouts



- “A callout is a standard or custom view that can appear with an annotation view”
- Standard callout displays the title, and can display additional info or images.
- The annotation object **needs a value set for its title property or the callout wont show.**
- Just set the canShowCallout property to true on the annotation view.
- Has left and right accessoryView properties that can be set to buttons, images, etc.
- mapView:calloutAccessoryControlTapped: tells you which annotation and callout was tapped.

Demo

Parse & Locations

- Parse is setup to allow you to easily incorporate locations into your data set.
- Locations are represented by the class PFGeoPoint.
- PFGeoPoints can be added to a PFObject like any other field, except each class can only have one PFGeoPoint field.

Parse & Locations

- So first off, how do you create PFGeoPoints?
- 3 initializers:

```
PFGeoPoint *seattle = [PFGeoPoint geoPointWithLatitude:47.60 longitude:122.33];
```

```
CLLocation *portlandLocation = [[CLLocation alloc] initWithLatitude:45.52 longitude:122.68];
PFGeoPoint *portland = [PFGeoPoint geoPointWithLocation:portlandLocation];
```

```
[PFGeoPoint geoPointForCurrentLocationInBackground:^void (PFGeoPoint *location, NSError *error) {

  if (error) {

  } else {
    //location acquired!
  }
}];
```

Location Queries

- Once you have a bunch of objects in your database with location information, you will probably want to start querying with that info.
- PFQuery has a restriction called ‘whereKey:nearGeoPoint:’ that lets you query on a location based field.
- In addition, there are variants of that requirement that allow you to set a limit on the miles/kilometers/radians of your query.
- Finally, you can simply query for all objects in a rectangular bounding box with ‘whereKey:withinGeoBoxFromSouthwest:toNortheast:

Location Caveats

- Remember, each PFObject class can only have one PFGeoPoint field.
- Using a nearGeoPoint constraint will limit results within 100 miles.

Demo