iOS Foundations II
Session 7

- TextFields
- UIImage and UIImageView
- Custom Table View Cell
- For loops

UITextField

- “A `UITextField` object is a control that displays editable text and sends an action message to a target object when the user presses the return button.”
- A text field can have a delegate to handle editing related notifications.
- When a user taps into a text field, the text field becomes the first responder and it brings the keyboard on screen.
- You are responsible for making sure the text field you are editing is not covered by the keyboard.
- **A UITextField is a subclass of UIView**

UITextField

- UITextfield has an intrinsic content size!
- It takes the size of the place holder text inside of it. There are two types of placeholder text:
  - Text: regular text that the textfield starts off with
  - Placeholder text: greyed out place holder text, used to tell the user what should go inside the field. Once a single character of real text is added to the text field, the place holder text vanishes.
- If you provide no text or placeholder text, the textfield intrinsically takes a very small width of 26 points.

Demo

UITextField Keyboard Dismissal

- By default the textfield does not dismiss the keyboard once the user hits return.
- The delegate of the textfield can be notified when the user hits return by implementing this method:
  - textFieldShouldReturn(textField : UITextField)
- In the implementation of this method, call resignFirstResponder() on the textfield passed in
- You can return true in this method, since pressing return in a textfield doesn't do anything, since it only supports one line.

Demo

UIImage

- A class used to display an image.
- UIImage's are immutable, you cannot change them after creation.
- So the only way to edit a UIImage is to make a copy of it.
- Since they are immutable, you are not allowed to access their underlying binary image data.
- Use the UIImagePNGRepresentation or UIImageJPEGRepresenation functions to get an NSData from a UIImage.

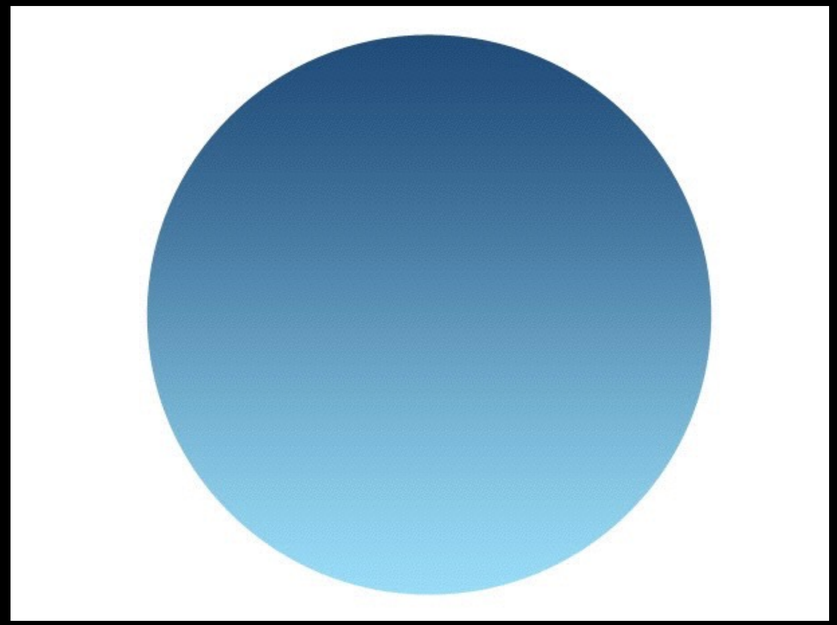
UIImage Supported Formats

Format	Filename extensions
Tagged Image File Format (TIFF)	`.tiff`, `.tif`
Joint Photographic Experts Group (JPEG)	`.jpg`, `.jpeg`
Graphic Interchange Format (GIF)	`.gif`
Portable Network Graphic (PNG)	`.png`
Windows Bitmap Format (DIB)	`.bmp`, `.BMPf`
Windows Icon Format	`.ico`
Windows Cursor	`.cur`
X Window System bitmap	`.xbm`

A note about images and memory: Images take up a lot memory, relative to things like strings and Ints (which is what most classes are made of).
The string ‘Brad’ takes up 4 bytes, where as the image above takes up 188,000 bytes.

JPG VS PNG

- JPGs and PNGs are the most common form of images you will deal with developing iOS applications.
- A rule of thumb: if its a high quality photo, it should be JPGs. Everything else should be a PNG.
- PNGs are typically larger than JPGs
- PNG's support transparencies:



JPG

PNG

How to get images in your app

- There are basically 3 ways an image can get inside your app:
    1. The image is placed inside the bundle (aka you drag into your Xcode project)
    2. The image is downloaded from a network resource
    3. The image is taken from the Camera/Photo Library

Getting the Image

- Images must be instances of the UIImage class in order for us to display them.
- Getting UIImages from images differs depending on how the image got inside your app:
  - If its inside your bundle, simply use this class method:

```
let myImage = UIImage(named: "seahawks")
```

  - If its coming from the network, you will need convert the data to an image:

```
let downloadedImage = UIImage(data: imageData)
```

  - If its coming from the camera/photo library, it will already be in UIImage form.

UIImageView

- View based container for displaying images(s)
- The image displayed is sized, scaled to fit, or positioned in the image view based on the imageView's contentMode.
- Apple recommends images displayed using the same imageView be the same size. If the sizes are super different, the system will create a brand new scale down image, which takes up more memory!

Demo

UIImageView Content Mode

```
typedef enum {
   UIViewContentModeScaleToFill,
   UIViewContentModeScaleAspectFit,
   UIViewContentModeScaleAspectFill,
   UIViewContentModeRedraw,
   UIViewContentModeCenter,
   UIViewContentModeTop,
   UIViewContentModeBottom,
   UIViewContentModeLeft,
   UIViewContentModeRight,
   UIViewContentModeTopLeft,
   UIViewContentModeTopRight,
   UIViewContentModeBottomLeft,
   UIViewContentModeBottomRight,
} UIViewContentMode;
```

- ScaleToFill: scale content to fit the size of itself by changing the aspect ratio if necessary
- ScaleAspectFit: scale content to fit the size of the view by maintaining aspect ratios. Any remaining area of the view's bounds is transparent.
- ScaleAspectFill:scale content to fill the size of the view. Some content maybe be clipped to fill the view's bounds.

UIImageView Content Mode

ScaleToFill:

AspectFit:

AspectFill

Custom UITableViewCell

TableViewCell Review

- UITableViewCell is a direct subclass of UIView.
- You can think of it as a regular view that contains a number of other views used to display information.
- The ‘Content View’ of a cell is the view that all content of a table view cell should be placed on. Think of it as the default super view of your cell. contentView itself is a read only property, you cant set it to be a different view.

TableViewCell Style

- Setting the style of an instance of UITableViewCell will expose certain interface objects on the cell.
- The default style exposes the default text label and optional image view.
- Right Detail exposes a right aligned detail text label on the right side of the cell in addition to the default text label.
- Left Detail exposes a left aligned detail text label on the right side of the cell in addition to the default text label.
- Subtitle exposes a left aligned label below the default text label.

Creating tableView Cells

- You can instantiate them in code with the initializer init(style: UITableViewCellStyle, reuseIdentifier: String?)
- But usually you will be setting them up in your storyboard or in a xib file.
- If they are in your storyboard, you just have to set their reuse identifier in the identity inspector, and then call dequeueReusableCellWithIdentifier() at the appropriate time.

Custom UITableViewCell

- Creating and laying out your own custom UITableViewCell is a relatively straight forward workflow:
  - Create a new class that is a subclass of UITableViewCell
  - In your storyboard, set your prototype tableview cell to be your new custom class
  - Drag out any interface elements you want onto your prototype cell, and then harness the badass power of auto layout to make it look amazing
  - Create outlets to each element in your custom class's implementation
  - Refactor your cellForRow:AtIndexPath: method on your tableview's datasource to use your new class

Demo

For loops

For Loops in Swift

- Swift provides all the familiar control flow from other C-like languages.
- For loops, while loops, if statements, and switch statements all fall under the category of control flow
- We will focus on for loops today, in particular the different ways Swift allows you to format and implement for loops.

For Loops in Swift

- Swift provides 2 types of for loops:
  - The regular ‘for loop’ performs a set of statements until a specific condition is met, typically by incrementing a counter each time the loop ends
  - The ‘for-in loop’ performs a set of statements for each item in a range,sequence,collection, or progression

Regular For Loop

- Here is an example of a regular for loop in Swift:

```
for var index = 0; index < 3; ++index {
    println("index is \(index)")
}
// index is 0
// index is 1
// index is 2
```

- So here is the general form of a for loop in Swift:

```
for initialization; condition; increment {
    statements
}
```

Demo

For-in Loop

- You use the for-in loop to iterate over collection of items, such as items in an array:

```
let names = ["Anna", "Alex", "Brian", "Jack"]
for name in names {
    println("Hello, \(name)!")
}
// Hello, Anna!
// Hello, Alex!
// Hello, Brian!
// Hello, Jack!
```

- You can even do it on a string!

```
for character in "Hello" {
    println(character)
}
```

Demo

Which one do I use?

- Most devs prefer to use the for-in loop vs the regular C style for loop. It looks cleaner and doesn't require any extra variable initialization or conditionals that may be prone to off-by-one errors.
- Of course, if you have no collection to loop through, you should the C style for loop.
- An advantage of the C style for loop is that you can keep track of the index you are on for each iteration of the loop.

Demo