iOS Foundations II
Session 6

- Homework Review
- UIImage
- UIImageView
- Camera Programming & UIImagePickerController
- Optionals

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.

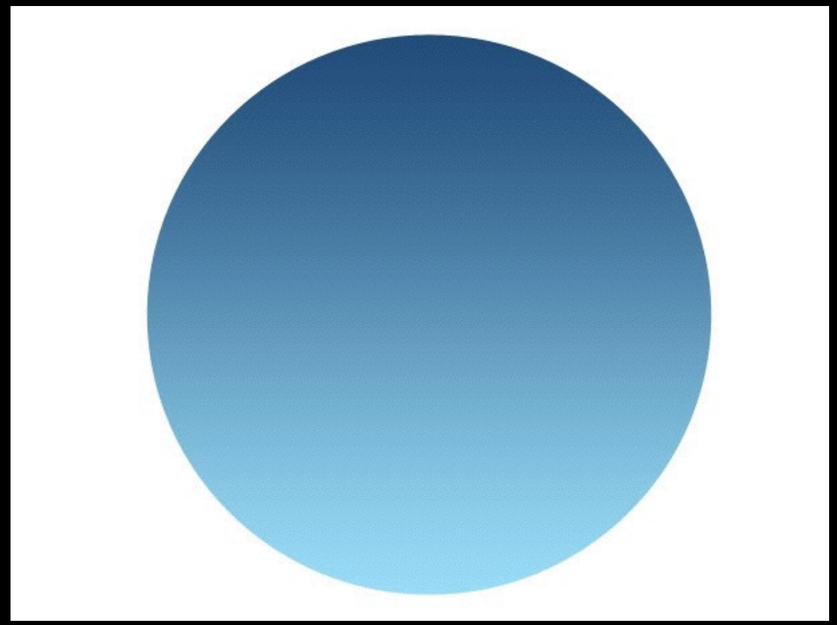
UIImage Supported Formats

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

Demo

Camera Programming

- 2 ways for interfacing with the camera in your app:
    1. UIImagePickerController (basic)
    2. AVFoundation Framework (advanced)
- We are going to use UIImagePickerController for our roster app.
- UIImagePickerController is just a view controller that you configure and then present.
- It has built in functionality for letting a user choose a photo by using the camera or letting them select a photo from their photo library.

UIImagePickerController

- The workflow of using UIImagePickerController is 3 steps:
    1. Instantiate, configure, and modally present the UIImagePickerController
    2. ImagePicker manages the user's interaction with the camera or photo library
    3. The system invokes your image picker controller delegate methods to handle the user being done with the picker.

UIImagePickerController Setup

- The first thing you have to account for is checking if the device has a camera.
- Use the isSourceTypeAvailable class method on UIImagePickerController to check if camera is available (it wont be available on the simulator)

UIImagePickerController Setup

- Next make sure something is setup to be the delegate of the picker. This is usually the view controller that is spawning the picker.
- The final step is to actually create the UIImagePicker with a sourceType of UIImagePickerControllerSourceTypeCamera.
- Media Types: Used to specify if the camera should be locked to photos, videos, or both.
- AllowsEditing property to set if the user is able to modify the photo in the picker after taking the photo.

UIImagePickerControllerDelegate

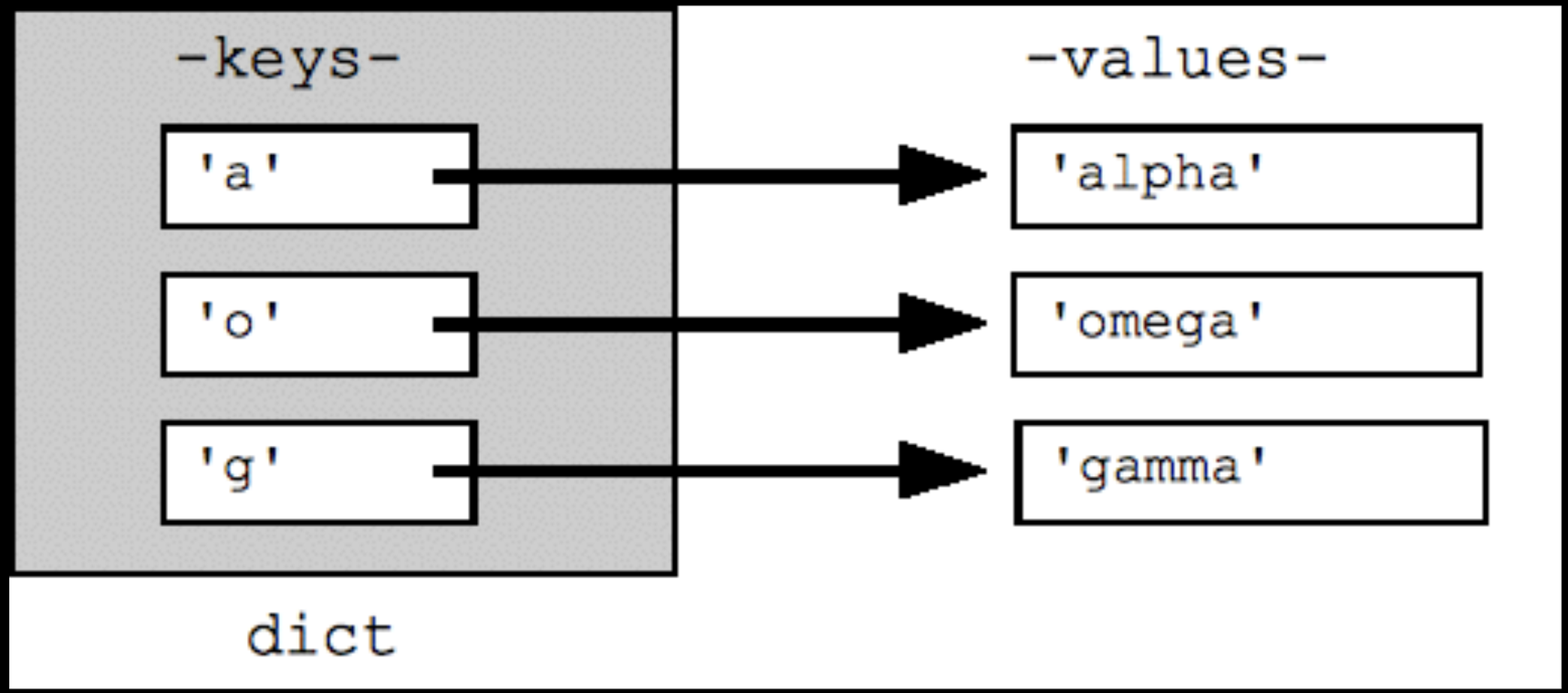
- The Delegate methods control what happens after the user is done using the picker. 2 big methods:
  1. imagePickerControllerDidCancel:
  2. imagePickerController:didFinishPickingMediaWithInfo:

Demo

Dictionaries

- Dictionaries are a collection type, like array.
- A dictionary keeps track of things in key-value pairings, as apposed to an array that keeps an index. Dictionaries are unordered in that regard.
- You store an object in a dictionary by calling setObject:ForKey:
- But you almost never use that long hand form, instead use the [] shorthand syntax, just like an array
- When you need to retrieve the object from the dictionary, you can call objectForKey: and provide the original key you set it with (use the short hand form instead)
- Dictionaries do not allow duplicate keys.

Dictionaries



Creating Dictionaries

- Just like with arrays, there is a literal syntax for creating Dictionaries:

```
var info = ["Year" : 2014]
```

- Setting a value in a dictionary also has a special shorthand syntax:

```
info["Month"] = 11
info["Day"] = 17
```

- As well as accessing a value:

```
var today = info["Day"]
```

Demo

Optionals

Optionals

- You use optionals in situations where a value may be absent.
- An optional says:
  - There is a value and it equals x

    OR

  - There isn’t a value at all.
- The concept of optionals does not exist in Objective-C or even C!

Marking an Optional

- A question mark at the end of a type indicates that the value it contains is optional, meaning it might contain a value or it might be empty:

```
class Person {

    var firstName : String?
```

Swift is strict!

- Swift does not allow you to leave properties in an undetermined state.
- They must be:
  - given a default value

    OR
  - a value set in the initializer

    OR
  - or marked as optional

Accessing a value inside an optional

To access the value inside of an optional variable, you must force unwrap it with the exclamation mark:

```
// return the first and last names in a string
func returnFullName() -> String {
    return "\(self.firstName!) \(self.lastName)"
}
```

You can also use a question mark to use whats called optional chaining.

Implicitly Unwrapped

Instead of marking an optional with a ?, using an exclamation point ! will mark it as implicit unwrapped:

```
class Person {

    var firstName : String!
```

This makes it so you don't have to unwrap this optional to access its value.
So basically you are saying “I’m making this an optional, but this thing will always have a value when I need to access it”

Optional Binding

- You can use **optional binding** to find out wether an optional contains a value, and if so, to make that value available as a temporary constant or variable that is unwrapped.
- The syntax of an optional binding:

```
if let constantName = someOptional {
    statements
}
```

using optional binding

```
func printTitleValue(value : String?) {

    if let title = value {
        println(title)
    }
}
```

not using optional binding

```
func printTitleValue(value : String?) {

    if value != nil {
        let title = value!
        println(title)
    }
}
```

Downcasting + Optional Binding

- It is very common to combine optional binding and downcasting
- Downcasting is used whenever a constant or variable of a certain type may actually refer to an instance of a subclass behind the scenes.
- When you think this is the case, you can use downcasting to attempt to cast the variable or constant to the subclass.
- There are two forms of down casting:
  - optional form: as?
  - forced form : as!

Demo