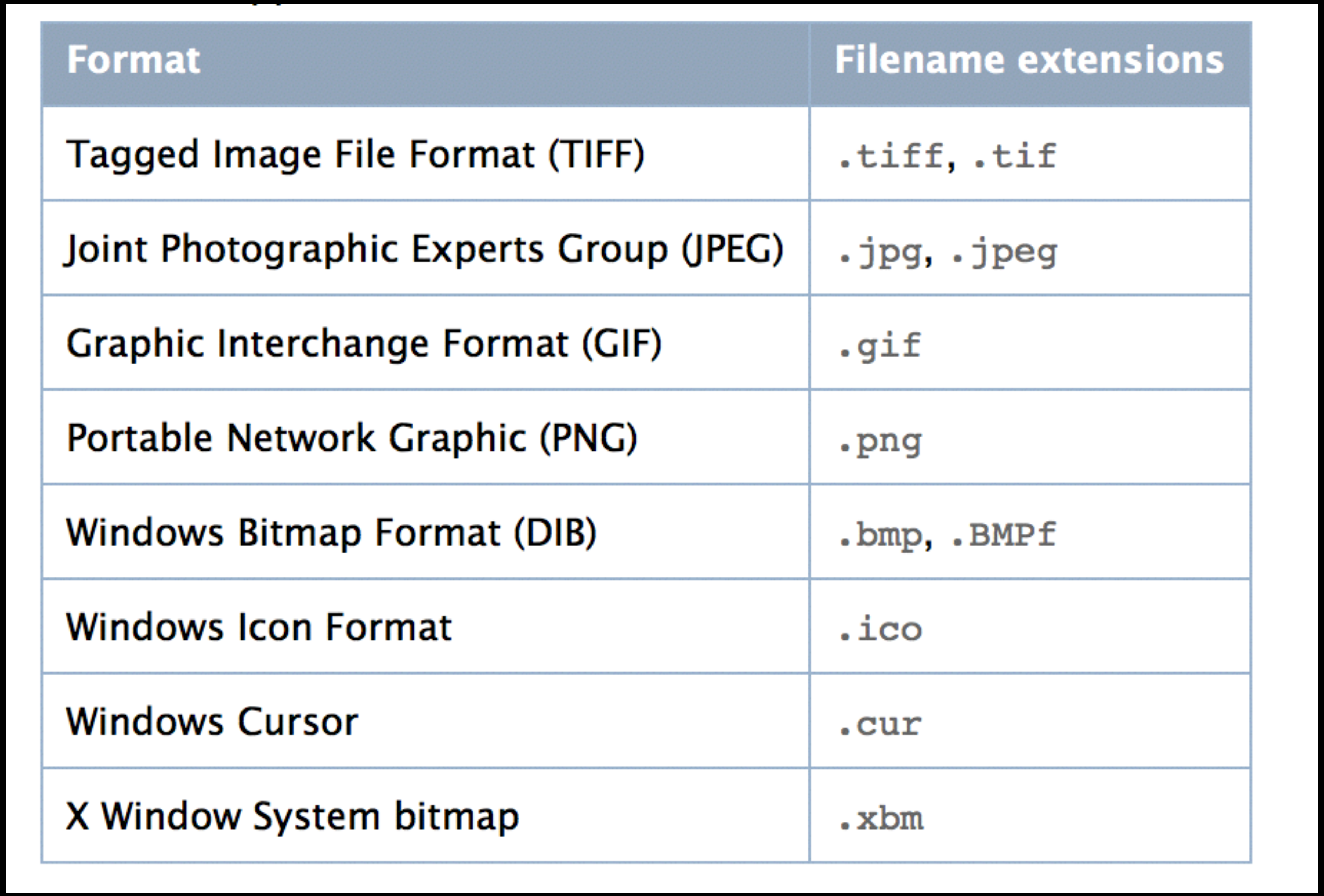
iOS Foundations II
Session 6

- Homework Review
- Reloading the table view
- UIImage
- UIImageView
- Camera Programming & UIImagePickerController

UIImage

- High-level way to display an image.
- UIImage’s are immutable, you cannot change them after creation. Which means they are thread safe.
- 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

UIImage Methods

- class func imageNamed(String) -> UIImage
- Returns an image object associated with a specified file name.
- Uses caching. If the image isn't in the cache, it then loads the image from the main bundle, caches it, and then returns the image.
- Since iOS 4, it is not no longer necessary to put .png into your string if its a png file. Still need .jpeg or .jpg though!
- If you know this image is only going to be used once, and don't need the caching, use the method imageWithContentsOfFile which doesn't cache. Saves memory.

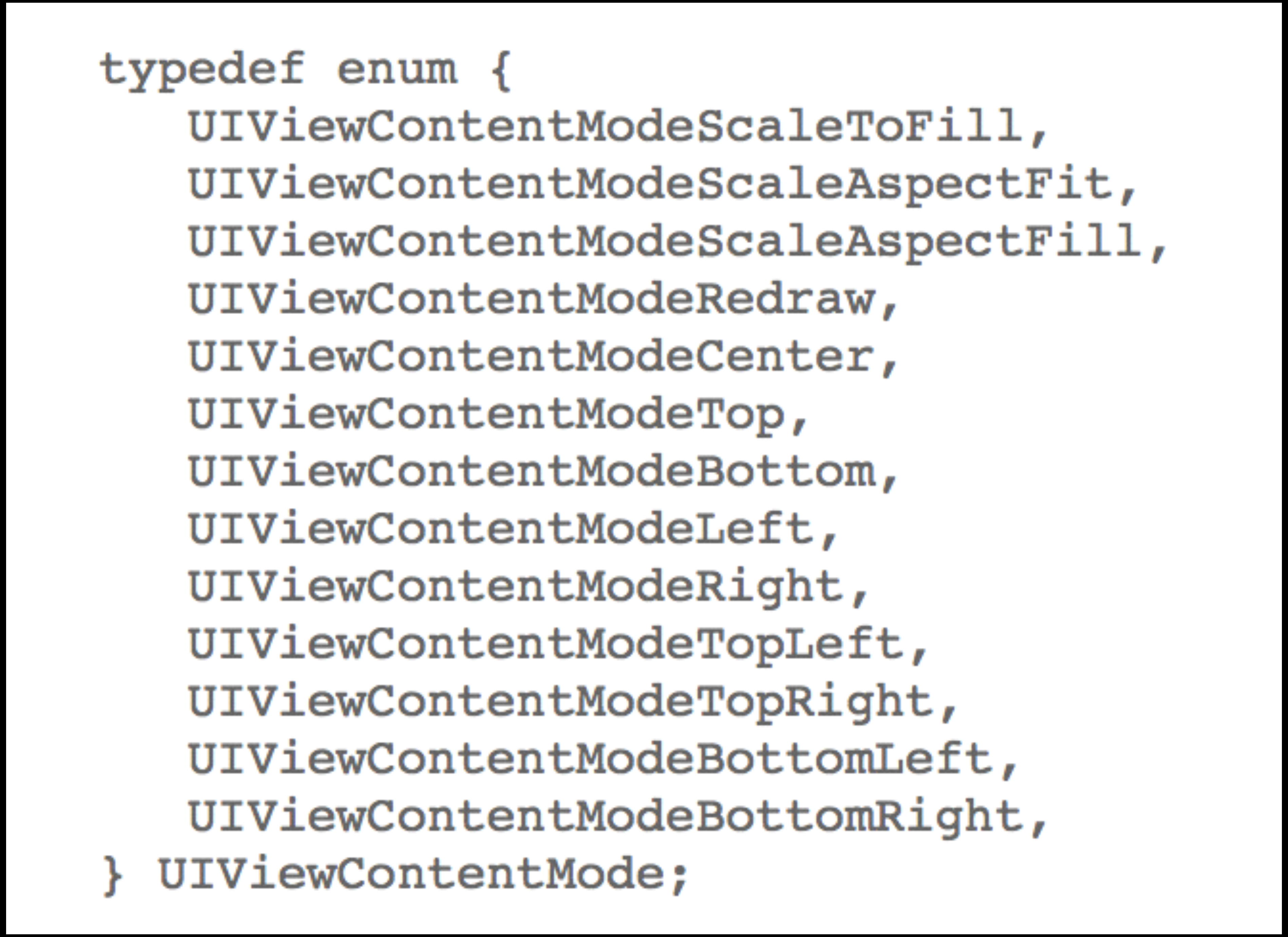
UIImage Methods

- class func imageWithData(NSData) ->UIImage
- The data passed in can be from a file or data you created.
- returns nil if it could not initialize the image from the specified data.
- no caching.

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!

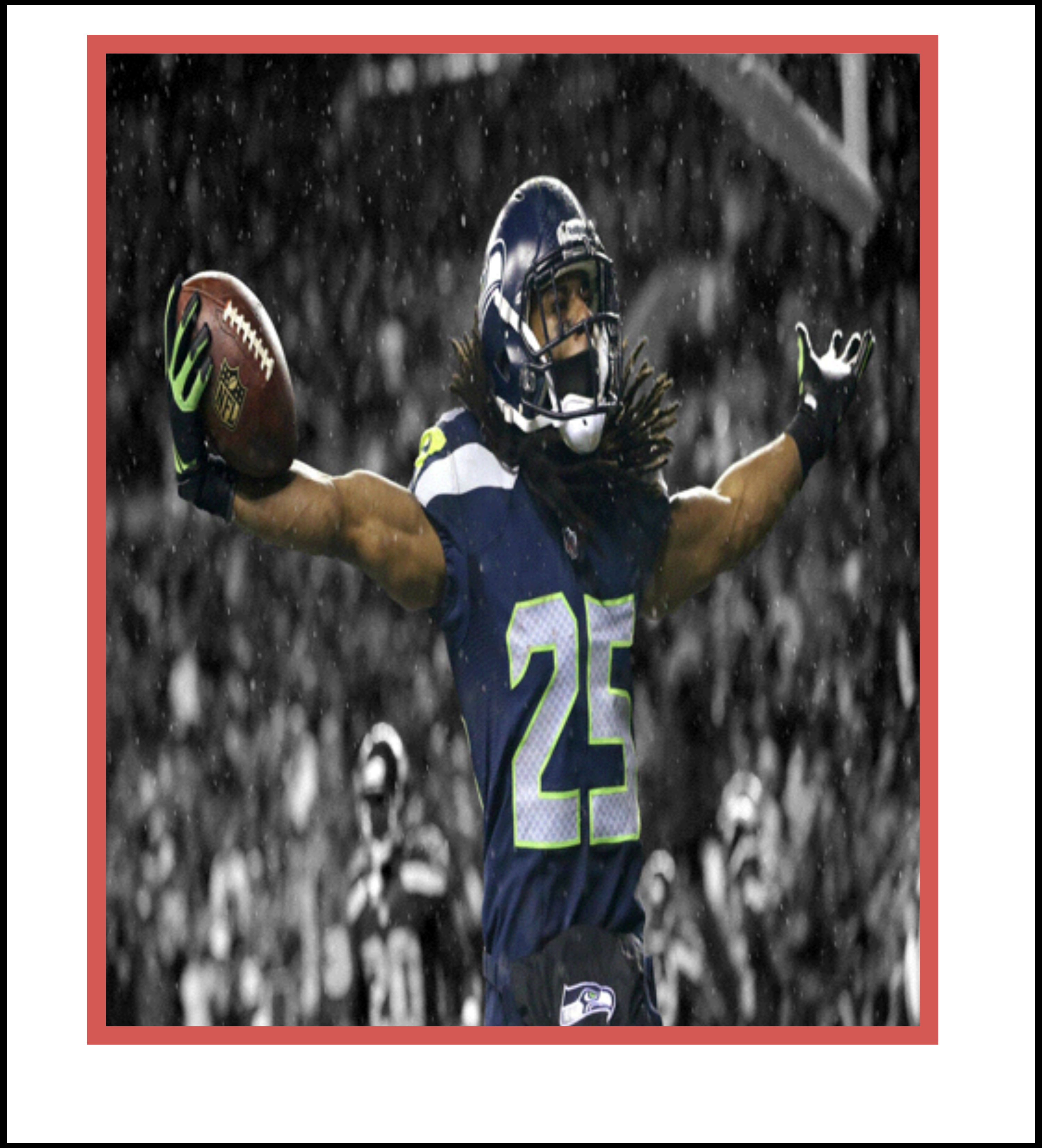
UIImageView Content Mode



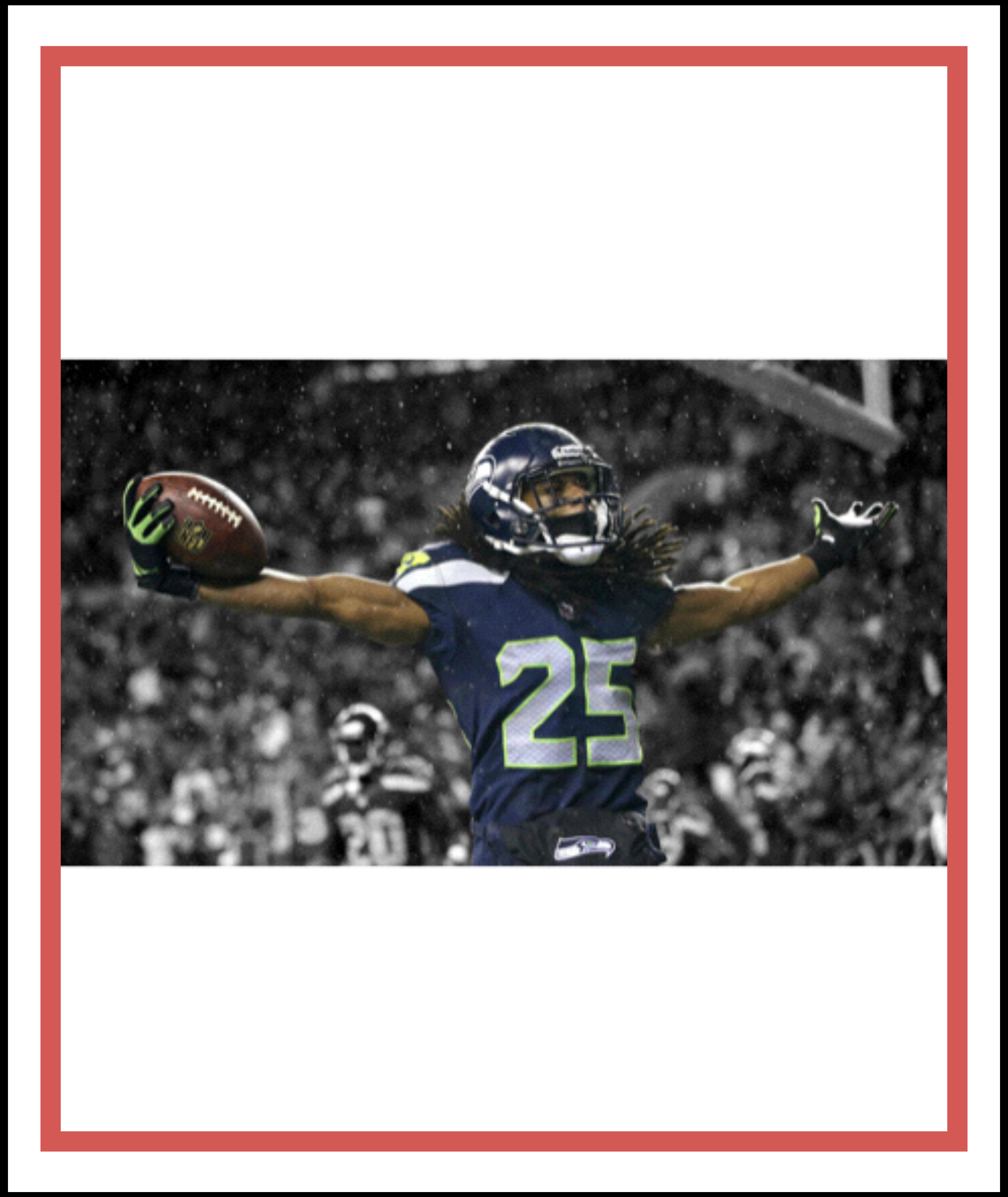
```
typedef enum {
   UIViewContentModeScaleToFill,
   UIViewContentModeScaleAspectFit,
   UIViewContentModeScaleAspectFill,
   UIViewContentModeRedraw,
   UIViewContentModeCenter,
   UIViewContentModeTop,
   UIViewContentModeBottom,
   UIViewContentModeLeft,
   UIViewContentModeRight,
   UIViewContentModeTopLeft,
   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.

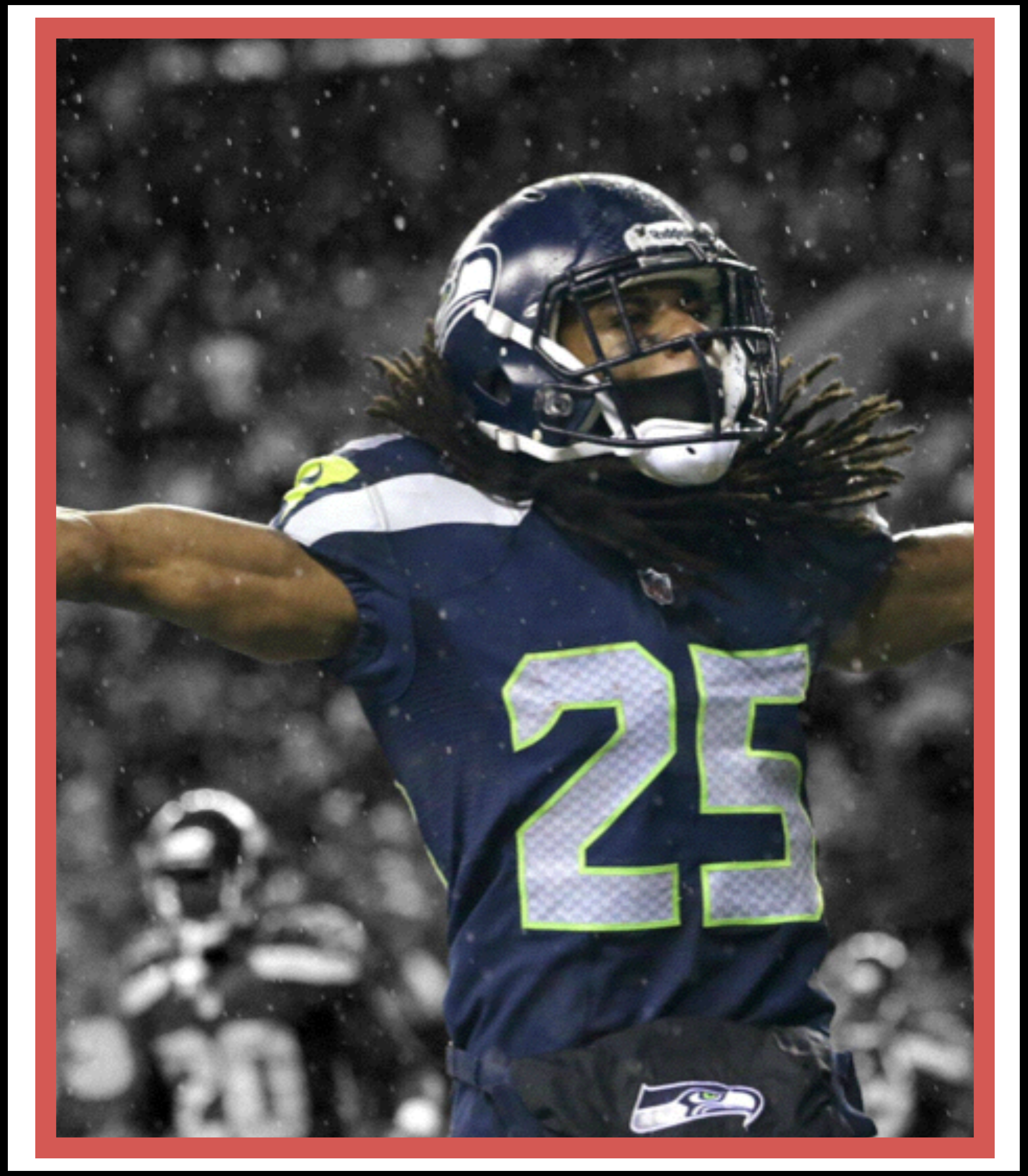
UIImageView Content Mode



ScaleToFill:



AspectFit:



AspectFill

UIImageView Optimization

- Pre-scale your images: if the imageView you using is very small, generate thumbnails of your images in a cache vs scaling large images to fit the small view.
- Disable alpha blending: Unless you are intentionally working transparency, you should mark your imageView's as Opaque.

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 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.
- If your app absolutely relies on a camera, add a UIRequiredDeviceCapabilities key in your info.plist
- 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:

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!

Swift is strict!

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

    OR

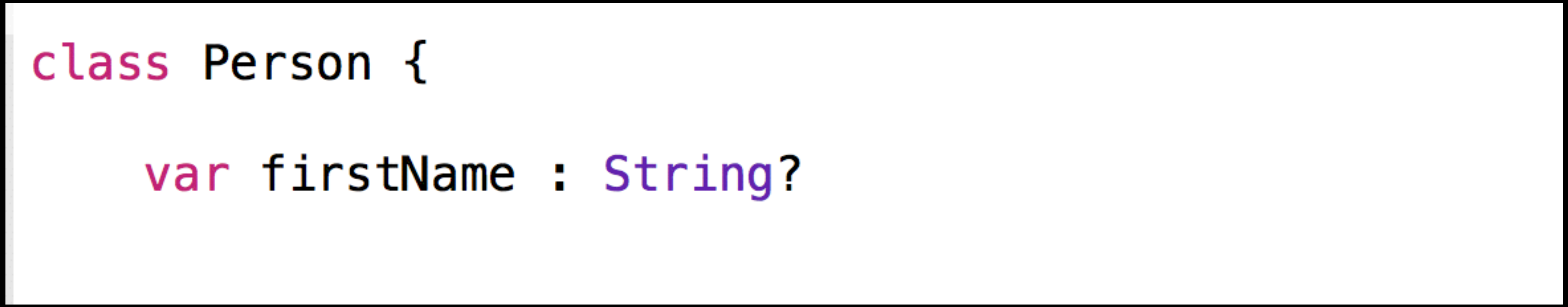
  - a value set in the initializer

    OR

  - or marked as optional

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:

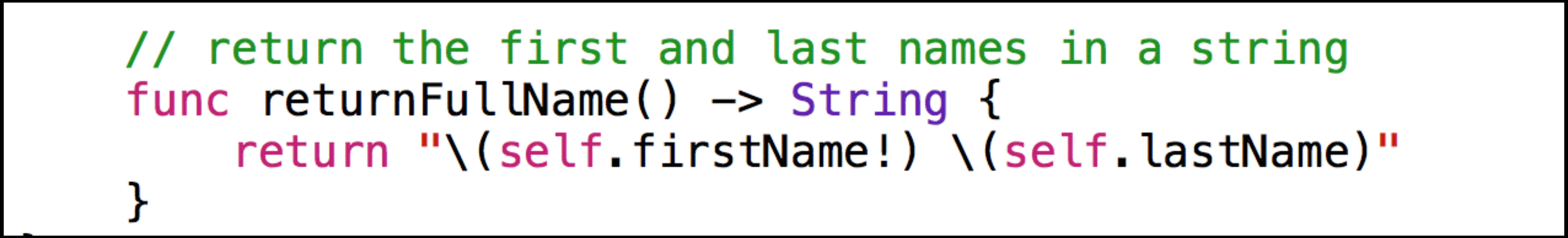


```
class Person {

    var firstName : String?
```

Accessing a value inside an optional

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

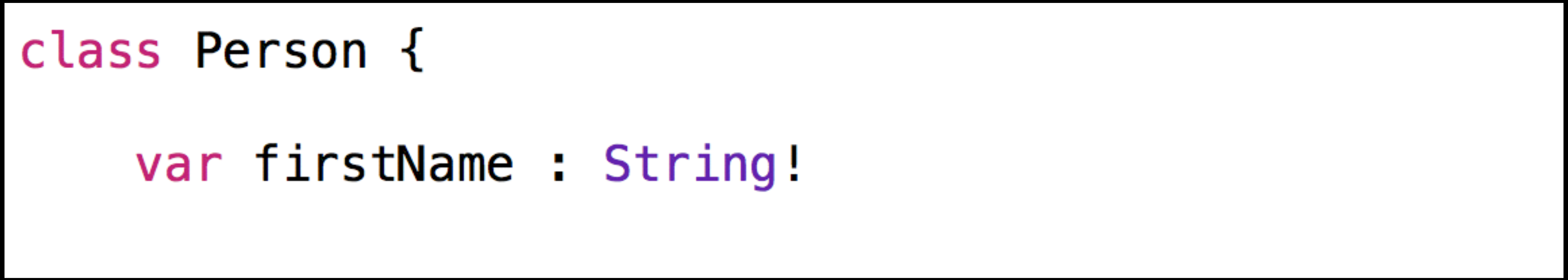


```
// return the first and last names in a string
func returnFullName() -> String {
    return "\(self.firstName!) \(self.lastName)"
}
```

You can also use a question mark to use whats called optional chaining. Optional chaining is a process for querying and calling properties,methods, and subscripts on an optional that might currently be nil.

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”

Demo