Saving Data on iOS

Archiving Overview

- What data to we want to save [An Array of Meals]
- Since the actual data consists of only Strings (Numeric Types are also ok), we can use Archiving, which preserves the structure of our data.
- If we had image files we'd save them separately, and save the *path* to the image in the food type.

Data Structures must be classes, subclassed from NSObject

```
import Foundation

struct Meal {
    var name: String
    var food: [Food]
}
```

```
import Foundation

class Meal: NSObject {
    var name: String
    var food: [Food]
}
```

Class 'Meal' has no initializers

```
init(name: String, food: [Food]) {
    self.name = name
    self.food = food
}
```

Description is a special field in NSObject.

- Replace description with food_description wherever it occurs.

NSCoding Protocol

- Objects which follow the NSCoding protocol can archive and unarchive themselves. This extends to Arrays of that kind of object, since Array conforms to NSCoding.

```
class Meal: NSObject, NSCoding {
```

Type 'Meal' does not conform to protocol 'NSCoding'

Do you want to add protocol stubs? Fix

NSCoding Functions

```
func encode(with aCoder: NSCoder) {
    aCoder.encode(name, forKey: "name")
    aCoder.encode(food, forKey: "food")
}
```

```
required convenience init?(coder aDecoder: NSCoder) {
    if let name = aDecoder.decodeObject(forKey: "name") as? String,
        let food = aDecoder.decodeObject(forKey: "food") as? [Food]
    {
        self.init(name: name, food: food)
    } else {
            return nil
    }
}
```

```
func encode(with aCoder: NSCoder) {
    aCoder.encode(name, forKey: "name")
    aCoder.encode(food_description, forKey: "food_description")
}

required convenience init?(coder aDecoder: NSCoder) {
    if let name = aDecoder.decodeObject(forKey: "name") as? String,
        let food_description = aDecoder.decodeObject(forKey: "food_description") as?
            String
    {
        self.init(name:name, food_description: food_description)
    }
        else {
        return nil
    }
```

Next we'll write a function to save and load 'meals'

- We'll put these in the Meal class, as static function
- Get the documents directory, and create a URL for the file

```
static let DocumentsDirectory = FileManager.default.urls(for: .documentDirectory,
    in: .userDomainMask).first!
```

```
static let ArchiveURL = DocumentsDirectory.appendingPathComponent("meals")
```

Save Function

```
static func saveToFile(meals:[Meal]){
    NSKeyedArchiver.archiveRootObject(meals, toFile: Meal.ArchiveURL.path)
}
```

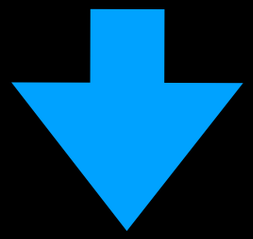
The root object is meals. You can have more than one - just use different URLS to save them.

Load Function

```
static func loadFromFile() -> [Meal]?  {
    return NSKeyedUnarchiver.unarchiveObject(withFile: Meal.ArchiveURL.path) as?
        [Meal]
}
```

Use your Save and Load Functions

```
override func viewDidLoad() {
    meals = initialMeals
}
```



```
override func viewDidLoad() {
    if let m = Meal.loadFromFile() {
        meals = m
    } else {
        meals = initialMeals
        Meal.saveToFile(meals: meals)
    }

}
```

Use your Save and Load Functions

```
@IBAction func unwindToFoodTableView(segue: UIStoryboardSegue){
    Meal.saveToFile(meals: meals)
    tableView.reloadData()
}
```

Other ways to save data on iOS

- Core Data - save data to SQLite Database (on device only)
- Cloudkit - save data to Apple Cloud Service - no login needed other than iCloud login, so can seem seamless.
- Other cloud services like Firebase. You'll need to use CocoaPods - a dependency manager for Xcode projects.
- Roll your own! I've used the WordPress JSON-API among other things.