# Basic Exercises Part 6.1 Parsing Data. JsonSerialization.

## Json 🡪 Serialization

* The JSON format was specified by Doublas Crockford, is a data interchange format.
* A JSON is simply a way of representing data independent of a platform.
* Is much easier to read and is also faster for computers to process because of its lightweight format. Is based on the object literal notation of JavaScript.
* The reason JSON is important is that many third parties such as Google, Yahoo, etc make web services that return JSON formatted data when you visit a URL with a specified query string. If you write your own web service, you’ll also probably find it really easy to convert your data to JSON when sending to another party.
* In all iOS old versions, you’ve used a third party library such as a Framework in order to parse the data. With iOS 5, needing to use a third party library for JSON parsing is obsolete. Apple added a JSON library in Cocoa.
* You can turn objects like NSString, NSNumber, NSArray and NSDictionary into JSON data and vice versa super easily. And of course no need to include external libraries – everything is done natively and super fast.

### **1.1 Create a new project**

Create a basic Single View or use Playgrounds. Let’s look at the simple case of converting an object or struct in our code to & from JSON. Then we’ll see how it might be more complicated when the JSON doesn’t match the objects & structs that we’re using in our code. We’ll use a .json file contains the JSON source.

### **1.2 JSON fake**

For testing purposes, create a fake API response in JSON format. Create a file as “.json” extension.

Create the file from next URL:

<https://cocoacasts.s3.amazonaws.com/building-a-weather-application-with-swift-3/decoding-json-data-in-swift-1/response.json>

### **1.3 Load sample response data**

Create a new file (File →New →File… →Swift → name it WeatherData →Next →Create).

Before we can parse the JSON data, we need to load it from response.json and deserialize it. If you want to parse JSON by hand rather than using **Codable**, iOS has a built-in alternative called **JSONSerialization** and it can convert a JSON string into a collection of dictionaries, arrays, strings and numbers in just a few lines of code.

import Foundation

// Fetch URL // normally this would be an API callback response

**let** url = Bundle.main.url(forResource: "response", withExtension: "json")!

// Load Data

**let** data = try! Data(contentsOf: url)

// Deserialize JSON

**let** JSON = try! JSONSerialization.jsonObject(with: data, options: [])

Remember to avoid force unwrapping.

Now print the JSON object. Review the result.

### **1.4 What’s happen here?**

We deserialize the Data instance using the JSONSerialization API. The API is easy to understand. We invoke jsonObject(with:options:), passing in a Data instance and a collection of options. Notice that we are ignore error handling for now by using the try! keyword. Once we have a solution, we can focus on error handling in the weather application. If no error is thrown, jsonObject(with:options:) returns an object of type Any.

### **1.5 Wrapping a possible error**

There are a couple of things that might confuse you there. First, because parsing JSON will fail if the JSON isn't valid, you need to use try/catch and have some sort of error handling. Second, you need to typecast the example JSON to be a dictionary of type **[String: Any]** so that you can start working with your JSON values. Third, you don't know for sure that any values exist inside the JSON, so you need to conditionally check for and unwrap one of the **“key values”** value.

**1.6 How Apple does**

Visit:

<https://developer.apple.com/swift/blog/?id=37>

**1.7 To dictionary and then to model**

The challenge is instead of just printing a string (long enough), request the information into a Dictionary. Print the dictionary. Try to create the module structure based on the response and in the dictionary. And remember to use MVC to structure your application.

**1.6 Switch language**

We always need to try the same behavior on Objective C.