JSON & URLSessions

Mobile Computing - iOS

Objectives

- Students will be able to:
    - describe the purpose of JSON
    - describe the syntax of valid JSON
    - use URLSessions to fetch data from a web service
    - read JSON into an app from a web service

JSON

- Pronounced "Jason" — at least according to the person who created it.

- An alternative to XML, an easy way to exchange data among programs

- JSON is comprised of:

  - **objects**, consisting of comma-delimited, colon-separated, **name/value** pairs, enclosed in { }

  - **arrays**, consisting of collections of comma delimited values, enclosed in [ ]

  - **values:** strings, numbers, objects, arrays, booleans or null

JSON

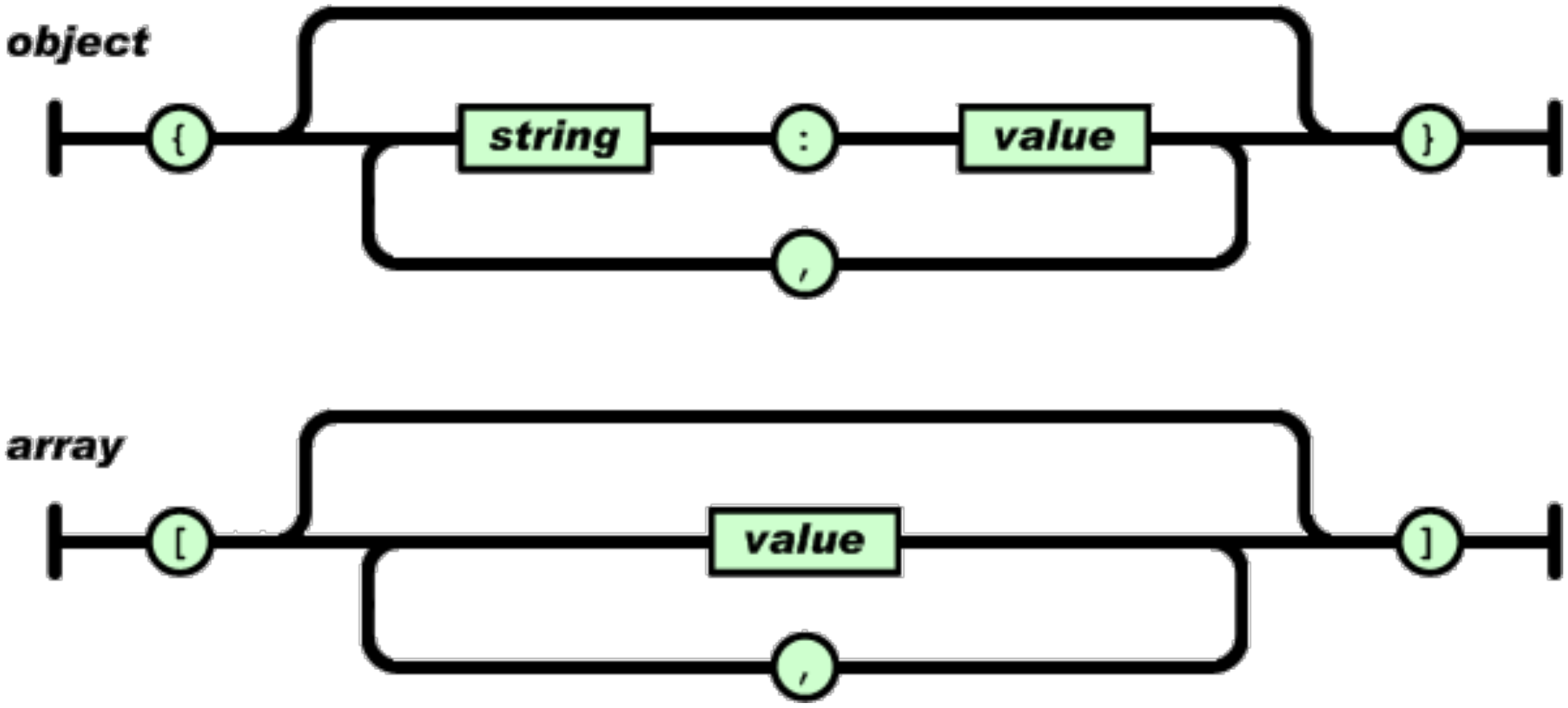
A set of key:value pairs, enclosed in { }

{"name":"Quentin", "age":25, "friends":["Penny", "Alice"]}

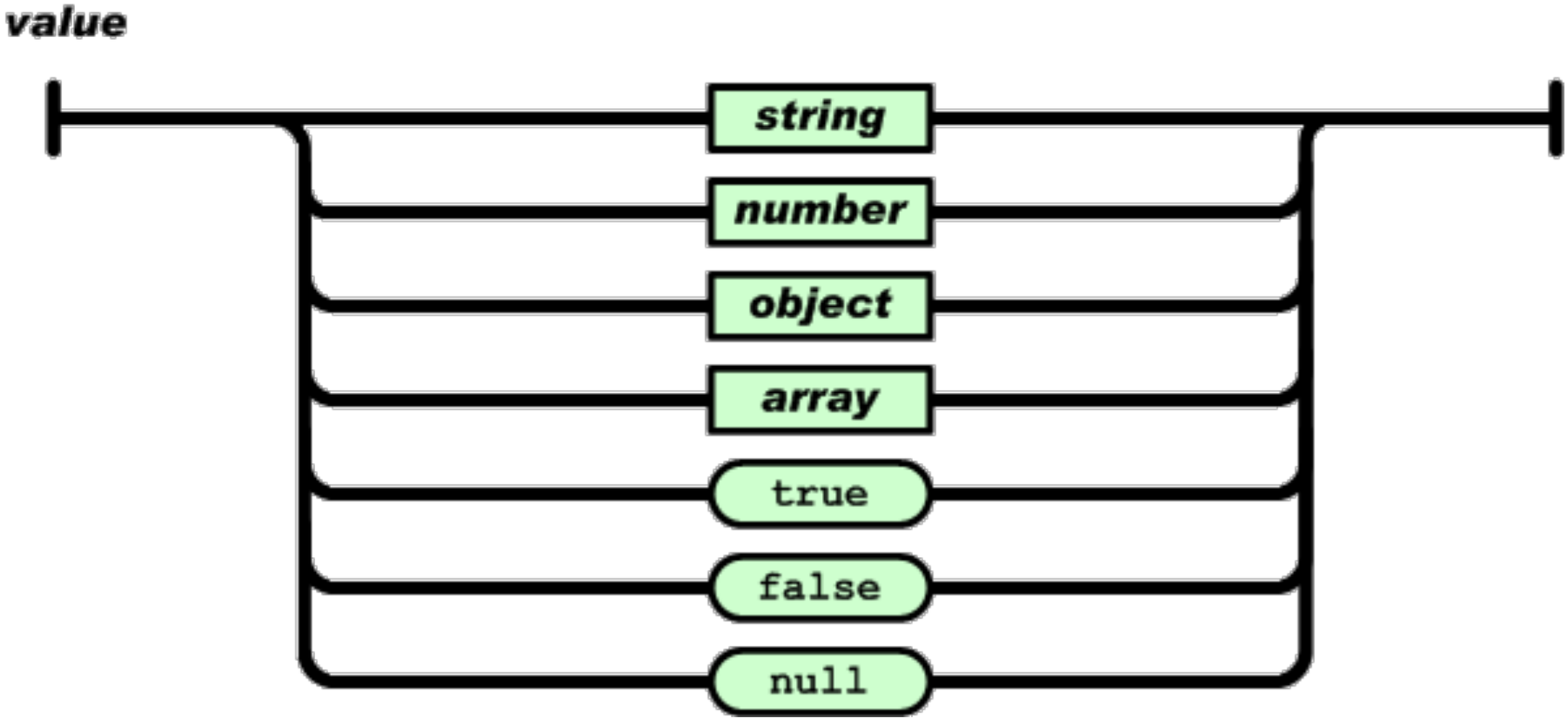
An array of Strings

- JSON arrays are represented as arrays in iOS (shocking!)
- JSON objects are represented as Dictionaries in iOS

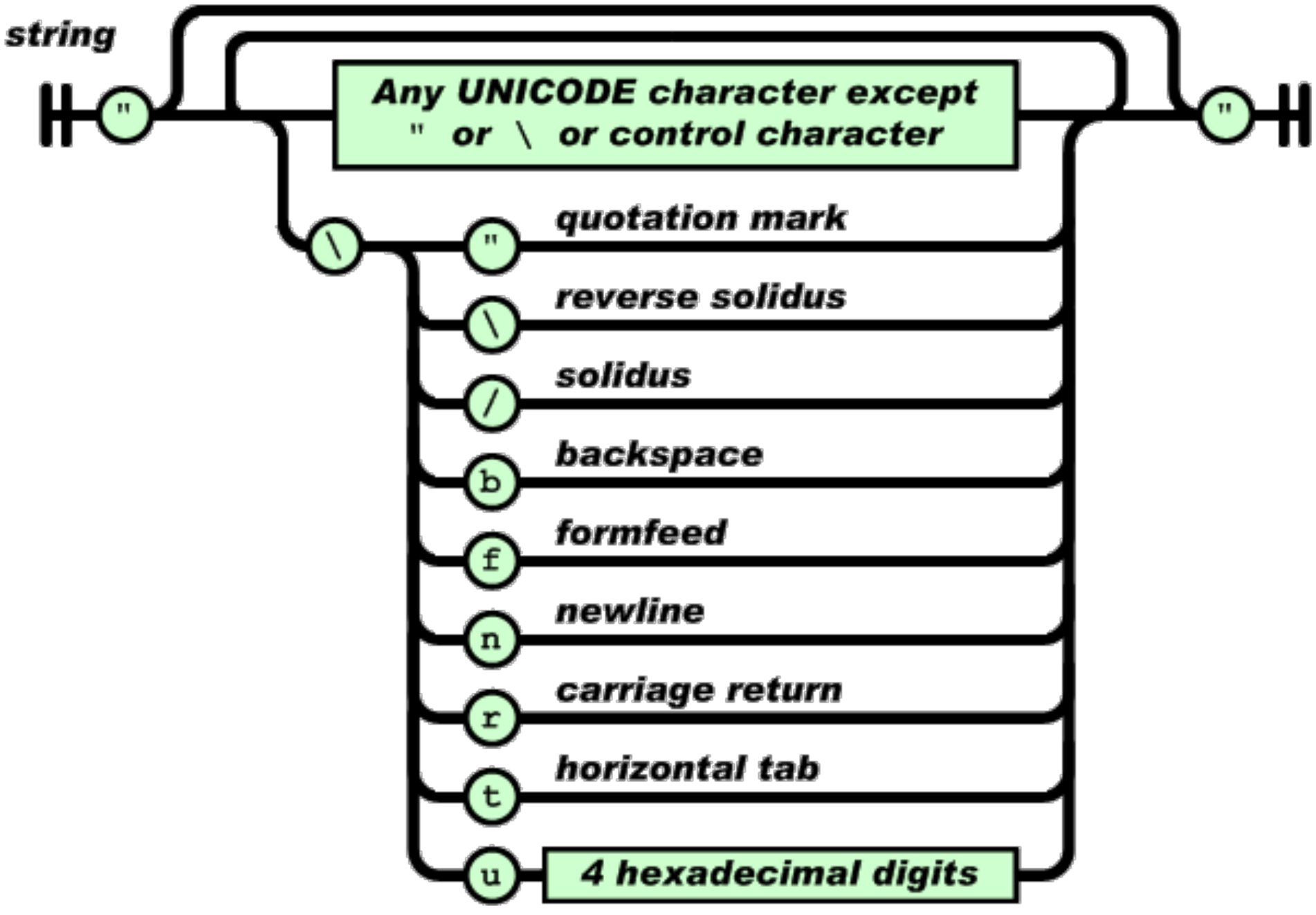
JSON: Objects and Arrays



JSON: Values



JSON: Strings



A JSON Example

{"adult":false,
"budget":9400000,
"title":"Finding Nemo",
"popularity":24.2,
"tagline":"There are 3.7 trillion fish in the ocean. They're looking for one.",
"genres":["animation","family"],
"production_countries":[
{"iso_3166_1":"US", "name":"United States"},
{"iso_3166_1":"CA", "name":"Canada"}]
}

https://api.themoviedb.org/3/movie/12?api_key={{API_KEY}}&language=en-US

Another JSON Example

```
[ {"name":"Afghanistan", "popn" 6373176,"inNATO":false},
  {"name":"Albania", "popn": 2876591, "inNATO":false},
  {"name":"Algeria", "popn": 42200000, "inNATO":false} ]
```

Another JSON Example


```
{"coord":{"lon":-94.87,"lat":40.35},
 "weather":[{"id":800,"main":"Clear","description":"clear
sky","icon":"01d"}],
 "base":"stations",
 "main":{"temp":288.39,"pressure":1017,
         "humidity":47,"temp_min":288.15,
         "temp_max":289.15},
 "visibility":16093,"wind":{"speed":5.7,"deg":
190},"clouds":{"all":1},"dt":1540829700,"sys":{"type":
1,"id":857,"message":0.0041,"country":"US","sunrise":
1540817193,"sunset":1540855147},"id":
420020184,"name":"Saint Joseph","cod":200}
```

Encoding and Decoding

- An app (or any program) has its own internal data structures (e.g., structs, classes, arrays)

- When transmitting data to other sources (a network, local storage, APIs/services), it needs to be **encoded** into a suitable format (e.g., JSON, XML, etc.) first

- When receiving data from other sources, it needs to be **decoded** so it can be stored in an internal structure

Encoding and Decoding

```
struct Planet {
 var name:String
 var mass:Double
 var distance:Double
}
```

Encode

```
{"name":"Jupiter",
 "mass":4E10,
 "distance":4.387}
```

Decode

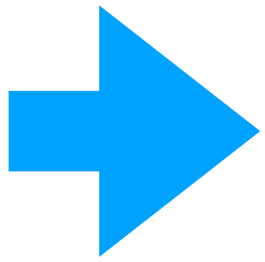
```
struct Planet {
 var name:String
 var mass:Double
 var distance:Double
}
```

Encoding and Decoding in Swift

- In Swift, any type (struct or class) that adheres to the **Codable** protocol can be
    - encoded into JSON using a **JSONEncoder**
    - decoded from JSON into a struct or class, using a **JSONDecoder**.
- When that happens, we say that the type is **codable** (lowercase)
- The Codable protocol has no required methods: if all of a type's properties are Codable, all you need to do is declare the type to be Codable and the compiler will smile approvingly

Making a Struct Codable

```
struct Planet {
    var name:String
    var mass:Double
    var distance:Double
}
```



```
struct Planet : Codable {
    var name:String
    var mass:Double
    var distance:Double
}
```

Now Planet is codable: easy as 🥧

- Since all of Planet's properties are Codable, all you need to do to make Planet Codable is declare it as such

- All the standard library types (Int, Double, Bool, String, etc.) are Codable; Date, URL, Data are too.

Types with Custom Types

```
struct Moon : Codable {
    var name:String
    var diameter:Double
}
```

```
struct Planet : Codable {
    var name:String
    var mass:Double
    var distance:Double
    var moons:[Moon]
}
```

- If your type (struct or class) contains a custom type, as long as that, too is codable, declaring your struct as Codable will still work

- Arrays of Codable objects are codable, so moons is codable (as are Strings and Doubles), so Planet too can be declared Codable

Encoding a struct into JSON

1. Make a struct
2. Make an encoder, an instance of JSONEncoder
3. Encode the struct using encode(), to get JSON.
    - encode() returns Data, a common format for storing all sorts of data. In this case, it will be a String (JSON).

Encoding a struct into JSON

```
struct Planet : Codable {
    var name:String
    var mass:Double
    var distance:Double
}

let earth = Planet(name: "Earth", mass: 2.503e17, distance: 1.0) // make a struct

let encoder = JSONEncoder()                                      // make an encoder

let encodedResult:Data = try encoder.encode(earth)               // encode the struct

print( String(data: encodedResult, encoding: .utf8)! )

// output: {"name":"Earth","mass":2.503e+17,"distance":1}
```

Encoding a [struct] into JSON

```
struct Planet : Codable {
    var name:String
    var mass:Double
    var distance:Double
}

let earth = Planet(name: "Earth", mass: 5.97e+24, distance: 149.6e6)
let mars = Planet(name:"Mars", mass:0.642e+24, distance:227.9e6)
let jupiter = Planet(name:"Jupiter", mass:1898e+24, distance: 778.6e6)

let encoder = JSONEncoder()

let encodedResult = try encoder.encode([earth,mars, jupiter])

print(String(data: encodedResult, encoding: .utf8)!)

/*
[
 {"name":"Earth","mass":5.970000000000003e+24,"distance":149600000},
 {"name":"Mars","mass":6.420000000000005e+23,"distance":227900000},
 {"name":"Jupiter","mass":1.8979999999999999e+27,"distance":778600000}
]
*/
```

An array can be encoded in the same fashion

Decoding JSON into a struct

1. Obtain some JSON data (we will see how to get this from a web service shortly)

2. Create a decoder, JSONDecoder()

3. Decode the data, using decode(), indicating what struct we expect to find in the JSON

Decoding JSON into a struct

```
let earth = Planet(name: "Earth", mass: 5.97e+24, distance: 149.6e6)
let encoder = JSONEncoder()
let earthJSON = try encoder.encode(earth) // we did this to get some JSON Data
// earthJSON ==  {"name":"Earth","mass":5.973e+24,"distance":1496}
```

```
// Get some JSON data (e.g., from a web service)

let decoder = JSONDecoder()                                  // make a decoder

let newEarth:Planet = try decoder.decode(Planet.self, from: earthJSON)
                                                             // decode the JSON

// newEarth == Planet(name: "Earth", mass: 5.97e+24, distance: 149600000.0)
```

Decoding JSON into a [struct]

```
let earth = Planet(name: "Earth", mass: 5.97e+24, distance: 149.6e6)
let mars = Planet(name:"Mars", mass:0.642e+24, distance:227.9e6)
let jupiter = Planet(name:"Jupiter", mass:1898e+24, distance: 778.6e6)
let encoder = JSONEncoder()
let planetaryJSON = try encoder.encode([earth, mars, jupiter]) // now we're using a [Planet]
```

```
let decoder = JSONDecoder()
let planets:[Planet] = try decoder.decode([Planet].self, from: planetaryJSON)
for planet in planets {
    print(planet)
}

/*
 Planet(name: "Earth", mass: 5.970000000000003e+24, distance: 149600000.0)
 Planet(name: "Mars", mass: 6.420000000000005e+23, distance: 227900000.0)
 Planet(name: "Jupiter", mass: 1.8979999999999999e+27, distance: 778600000.0)
 */
```

URLSessions

- Web services make data available in JSON format.
- In order to download that data, we need to use **URLSession**, an "API for downloading content". It supports data, file, ftp, http and https protocols.
- The URLSession class is extensive / complex. We only look at the basics: read the docs for details.

Obtaining JSON Data with a URLSession

- After creating an URLSession, add a task(s) to be carried out to that session.
    - URLSessionDataTask (to download to a Data* object, stored in memory)
    - URLSessionDownloadTask (to download to a file)
    - URLSessionUploadTask (for uploading a file)
- Each task starts out in a suspended session: use **resume()** to download it.

*Mentioned earlier, Data wraps an arbitrary collection of bytes into an "opaque" object.

Using an URLSession

1. Obtain a URLSession using shared

```
let urlSession = URLSession.shared
```

2. Create a dataTask and attach it to the URLSession, with this URLSession method:

```
dataTask(with url: URL,
completionHandler: (Data?, URLResponse?, Error?) -> Void)
                                                  -> URLSessionDataTask
```

3. Tell the dataTask to resume()

Is this a line of code or an API? How can you tell?

- When the task completes, the completion handler will be called.
- The Data parameter will contain the information downloaded from the URL, ready to decode 😍

Time to Visit the ISS (International Space Station)

http://api.open-notify.org/iss-now.json


```
{"message": "success",
 "timestamp": 1509385248,
 "iss_position": {"longitude": "-64.9925", "latitude": "26.3948"}}
```


```
struct IssLocation : Codable {
        var message:String
        var timestamp:Int
        var iss_position:Position
}

struct Position : Codable {
    var latitude:String
    var longitude:String
    }
```

We build the structs based on the JSON. The structs' structure must reflect that of the

ISS_Location Example

Download the example

```
func fetchISSData(){
    let urlSession = URLSession.shared
    let url = URL(string: "http://api.open-notify.org/iss-now.json")!
    urlSession.dataTask(with: url, completionHandler: showISSData).resume()
}

func showISSData(data:Data?, urlResponse:URLResponse?, error:Error?){
    do {
        let decoder = JSONDecoder()
        let location = try decoder.decode(IssLocation.self, from: data!)
        print(location)
    }catch {
        print(error)
    }
}
```

```
{"message": "success", "timestamp": 1509385248,  "iss_position":
{"longitude": "-64.9925", "latitude": "26.3948"}}
```

An Example with an Array

Download the example

```
var planets: [Planet] = []

@IBAction func fetchPlanets(sender:Any) {
        let urlSession = URLSession.shared
        let url = URL(string: "https://.../planets.json")
        urlSession.dataTask(with: url!, completionHandler: displayPlanetsInTableView).resume()
    }

func displayPlanetsInTableView(data:Data?, urlResponse:URLResponse?, error:Error?)->Void {
        do {
            let decoder:JSONDecoder = JSONDecoder()
            planets = try decoder.decode([Planet].self, from: data!)
            DispatchQueue.main.async() { self.planetTV.reloadData() }
        } catch {
            print(error)
        }
    }
```

planets.json

```
[
 {"name":"Earth","mass":5.97000000000000e+24,"distance":149600000},
 {"name":"Mars","mass":6.42000000000000e+23,"distance":227900000},
 {"name":"Jupiter","mass":1.89799999999999e+27,"distance":778600000}
]
```

```
struct Planet : Codable {
    var name:String
    var mass:Double
    var distance:Double
}
```

Techy Aside: JSON and Code Name Mismatch

- What happens if the JSON and your code do not have the same names?

- You can't change the JSON, but you can change your code to include an enum that must be named **CodingKeys**, in which the values match your code, and the rawValues match the JSON .

- For instance, suppose that you wished to use planetName, massKG and distance in your code, while the JSON used name, mass and distance.

```
struct Planet : Codable {
    var planetaryName:String
    var massKG:Double
    var distance:Double

    enum CodingKeys : String, CodingKey {
        case planetaryName = "name", massKG = "mass", distance = "distance"
    }
}
```

Techy Aside: Asynchronous Actions in a Playground

- If we try to run the previous code in a playground, the playground will finish before the callback is executed. We need to tell the playground that it should wait for the response.

```
import PlaygroundSupport
PlaygroundPage.current.needsIndefiniteExecution = true
```

JSONSerialization

- There is *another* way to parse JSON data that avoids having to build a struct first

- Invoke **jsonObject(with:options:)**, passing it the data returned from a URLSession.

- It will return either a **Dictionary** or an **Array**, depending on the JSON being processed: JSON *objects* get returned as *dictionaries*, each JSON name corresponding to a key in the dictionary; JSON arrays get returned as arrays.

- Since the return type is Any, downcast to get to the actual value

```
class func jsonObject(with data: Data,
                      options opt: JSONSerialization.ReadingOptions = []) throws -> Any
```

JSONSerialization

- Why might you want to do this? You might be:

  1.dealing with legacy code

  2.writing in Swift 3 and below (but hopefully not)

  3.looking at a lengthy/complex JSON object / array, making the struct difficult to construct

```
class func jsonObject(with data: Data,
                      options opt: JSONSerialization.ReadingOptions = []) throws -> Any
```

A (More) Complex Struct


```
{
 "coord":{
   "lon":-94.87,
   "lat":40.35
 },
 "weather":[
 {
   "id":800,
   "main":"Clear",
   "description":"clear sky",
   "icon":"01d"
 }
 ],
 "base":"stations",
 "main":{
   "temp":79.72,
   "pressure":1018,
   "humidity":57,
   "temp_min":78.8,
   "temp_max":80.6
 },
 "visibility":16093,
  "wind":{
    "speed":10.29,
    "deg":170
  },
  "clouds":{
  "all":1
  },
  "dt":1503605700,
  "sys":{
  "type":1,
  "id":857,
  "message":0.0217,
  "country":"US",
  "sunrise":1503574818,
  "sunset":1503622918
  },
  "id":5056172,
  "name":"Maryville",
  "cod":200
  }
```


What type is the overall JSON?
What type is coord?
What type is weather?
What type is base?
What type is main?

A (More) Complex Struct


```
{
 "coord":{
   "lon":-94.87,
   "lat":40.35
 },
 "weather":[
 {
   "id":800,
   "main":"Clear",
   "description":"clear sky",
   "icon":"01d"
}
],
"base":"stations",
"main":{
  "temp":79.72,
  "pressure":1018,
  "humidity":57,
  "temp_min":78.8,
  "temp_max":80.6
},
"visibility":16093,
 "wind":{
   "speed":10.29,
   "deg":170
 },
 "clouds":{
 "all":1
 },
 "dt":1503605700,
 "sys":{
 "type":1,
 "id":857,
 "message":0.0217,
 "country":"US",
 "sunrise":1503574818,
 "sunset":1503622918
 },
 "id":5056172,
 "name":"Maryville",
 "cod":200
 }
```


What type is the overall JSON? - object
What type is coord? - object with 2 values
What type is weather? - array with 1 object with 4 values
What type is base? - string
What type is main? - object with 5 values

A (More) Complex Struct Example ...

```
func displayTemperature(data:Data?, urlResponse:URLResponse?, error:Error?)->Void {
  var weatherRecord:[String:Any]!
  var mainRecord:[String:Any]!
  var temperature:Double!
  do {
    try weatherRecord = JSONSerialization.jsonObject(with: data!,
                                options: .allowFragments) as? [String:Any]
    if weatherRecord != nil {
      mainRecord = weatherRecord["main"]! as! [String:Any] // not [String:Double]
      temperature = mainRecord["temp"]! as! Double
                                }
    }catch {
      print(error)
  }
}
```

1

2

2

1. since the entire JSON is an object
2. since weatherRecord["main" is also an object
3. since mainRecord["temp"] is a Double

```
let openWeatherMapAPIKey = "xxxx"

let openWeatherMapURL = "https://api.openweathermap.org/data/2.5/weather?
id=5056172&appid=xxxx&units=imperial"

// called to start the temperature fetching process
func fetchTemperature() -> Void {

  let urlSession = URLSession.shared
  let url = URL(string: openWeatherMapURL)
  urlSession.dataTask(with: url!, completionHandler: displayTemperature).resume()
}
```



```
{
 "coord":{
 "lon":-94.87,
 "lat":40.35
 },
 "weather":[
 {
 "id":800,
 "main":"Clear",
 "description":"clear sky",
 "icon":"01d"
 }
 ],
 "base":"stations",
 "main":{
 "temp":79.72,
 "pressure":1018,
 "humidity":57,
 "temp_min":78.8,
 "temp_max":80.6
 },
 "visibility":16093,
 "wind":{
 "speed":10.29,
 "deg":170
 },
 "clouds":{
 "all":1
 },
 "dt":1503605700,
 "sys":{
 "type":1,
 "id":857,
 "message":0.0217,
 "country":"US",
 "sunrise":1503574818,
 "sunset":1503622918
 },
 "id":5056172,
 "name":"Maryville",
 "cod":200
 }
```

Example: Parsing a JSON Array of Objects

- In the example at right, the outermost structure is a 5-element *array*, so it would be typecast as [Any]
- Each element is an object, so it would be parsed as a dictionary, [String:Any].
- Collectively then, we can parse the JSON as [[String:Any]]
- The properties of each object are respectively Int, Int and String, and would be parsed as such.
- Once we have those available, we can create Swift structs (or classes) out of them, store them in an array, and use them however they are needed.
- See the code on the next 2 slides

```
[{
   "Id": 114,
   "DirectorateID": 45,
   "Name": "Police Unit"
 },
 {
   "Id": 115,
   "DirectorateID": 46,
   "Name": "Post Office"
},
{
  "Id": 116,
  "DirectorateID": 47,
  "Name": "RedWeb"
},
{
  "Id": 117,
  "DirectorateID": 48,
  "Name": "RR Donnelley"
},
{
  "Id": 118,
  "DirectorateID": 49,
  "Name": "Sodexo"
}]
```

Parsing a JSON Array of Objects - Example

```
let scottishParliamentUrl = "https://data.parliament.scot/api/departments"

func fetchScottishServices() -> Void {
    let urlSession = URLSession.shared
    let url = URL(string: scottishParliamentUrl)
    urlSession.dataTask(with: url!, completionHandler: displayDepts).resume()
}
```

```
 struct ScottishDepartment {
     var id:Int?
     var directorateId:Int?
     var name:String?
 }
```

Parsing a JSON Array of Objects - Example

```
func displayDepts(data:Data?, urlResponse:URLResponse?, error:Error?)->Void {
  var depts:[[String:Any]]
  var department:[String:Any]!
  var scotDepts:[ScottishDepartment] = []
  do {
  // the JSON is an array of objects, so typecast as [[String:Any]]
    try depts = JSONSerialization.jsonObject(with:data!, options: .allowFragments) as!
                                                        [[String:Any]]
    // Each element of scottishDepartments is an object, so parse it as [String:Any]
    for i in 0 ..< depts.count {
      department = depts[i]

     // The department properties are Int, Int and String, so typecast appropriately:
     let id = department["Id"] as? Int
     let directorateId = department["DirectorateID"] as? Int
     let name = department["Name"] as? String
     scotDepts.append(ScottishDepartment(id:id, directorateId:directorateId, name:name))
    }
    // Now scotDepts can be sent to a TVC via a notification, printed, &c.
    NotificationCenter.default.post(name: NSNotification.Name("scotland"), object: scotDepts)
    for dept in scotDepts{
       print(dept)
    }
  }catch {
    print(error)
  }
}
```

Normally, there is no reason to write dept instead of department
But we are running out of horizontal space, hence this deviation

Example: Parsing a JSON Object with An Array

- In the example at right, the outermost structure is an object, so it would be typecast as a dictionary [String:Any]

- The object's property names (people, message, number), used as keys into that dictionary, would allow us to access the array, string and integer, respectively. We would have to typecast to get to these.

- See the code on the next slide, or download the playground here

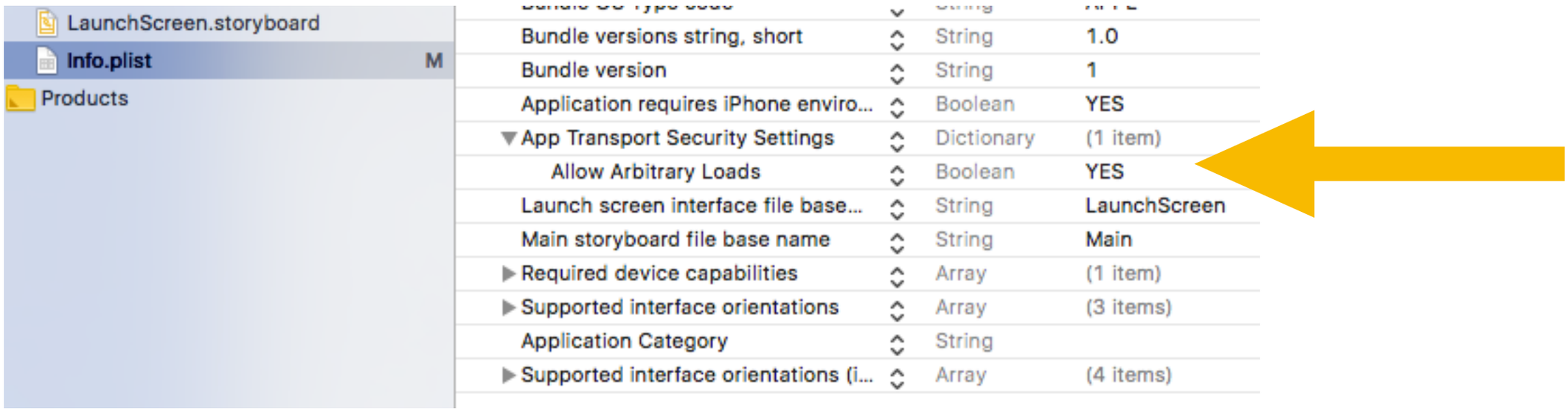
{"people": [{"name": "Sergey Prokopyev", "craft": "ISS"}, {"name": "Alexander Gerst", "craft": "ISS"}, {"name": "Serena Aunon-Chancellor", "craft": "ISS"}], "message": "success", "number": 3}

```swift
let astronautsUrl = "http://api.open-notify.org/astros.json"


func displayAstronauts(data:Data?, urlResponse:URLResponse?, error:Error?)->Void {
    var astronautsJson:[String:Any]!

    do {
        // outermost structure is an object, so typecast as [String:Any]
        try astronautsJson = JSONSerialization.jsonObject(with: data!,
                                                options: .allowFragments) as? [String:Any]
        // use JSON property names as keys into the dictionary we just fetched
        // to get to the info we need. We will need to typecast
        let message = astronautsJson["message"] as! String
        let numberOfAstronauts = astronautsJson["number"] as! Int
        print(message,numberOfAstronauts)
        let people = astronautsJson["people"] as! [Any] // since people is an array
        for i in 0 ..< people.count {
            let astronaut = people[i] as! [String:Any] // since the elements are objects,
                                                       // typecast as [String:Any]
            let name = astronaut["name"] as! String
            let craft = astronaut["craft"] as! String
            print(name,craft)
        }

    }catch {
        print(error)
    }
 }

func fetchAstronauts() -> Void {

    let urlSession = URLSession.shared
    let url = URL(string: astronautsUrl)
    urlSession.dataTask(with: url!, completionHandler: displayAstronauts).resume()
 }
```

http v. https

- There is a problem using http: it is a potential security risk, so it is blocked by default in apps.

- https://stackoverflow.com/questions/31254725/transport-security-has-blocked-a-cleartext-http



Another JSON ICE

1. Get an App ID key for openexchangerates.org
2. Visit https://openexchangerates.org/api/latest.json?app_id=YOUR_APP_ID
3. What sort of overall JSON is returned? An object or array?
4. What value is associated with the key "disclaimer"? with "rates"?
5. Download JSON The Argonaut
6. Place your App ID key in the openExchangeRatesAPI in CurrencyFetcher
7. Run the program
8. Remember that a JSON object is returned as a [String:Any] dictionary
9. What type is exchangeRates["disclaimer"]?
10. What type is exchangeRates["rates"]?
11. Write a statement to typecast (using as!) exchangeRates["rates"] as the type you identified in step 10: store it in a constant, allRates
12. From allRates, how will you get the conversion rate for, say, a EUR? or INR? or any other currency? Store it in a variable, euroRate or rupeeRate


```
{
  "disclaimer": "Usage",
  "license": "Omitted",
  "timestamp": 1509483600,
  "base": "USD",
  "rates": {
    "AED": 3.6728,
    "AFN": 68.305,
    "ALL": 114.75,
    "AMD": 482.71,
    "ANG": 1.785454,

     ...

    "WST": 2.529573,
    "XAF": 563.240161,
    "XAG": 0.05977306,
    "XAU": 0.00078676,
    "XCD": 2.70255,
    "XDR": 0.711903,
    "XOF": 563.240161,
    "XPD": 0.00101499,
    "XPF": 102.464689,
    "XPT": 0.00108697,
    "YER": 250.25,
    "ZAR": 14.131192,
    "ZMW": 10.015,
    "ZWL": 322.355011
  }
}
```

Another JSON ICE - Solution

1.Get an API key for openexchangerates.org

2.Visit https://openexchangerates.org/api/latest.json?app_id=YOUR_API_KEY

3.What sort of JSON is returned? An object or array?

4.What value is associated with the key "disclaimer"? with "rates"?

5.Download JSON The Argonaut

6.Place your API key in the openExchangeRatesAPI in DataFetcher

7.Run the program

8.Remember that a JSON object is returned as a [String:Any] dictionary

9.What type is exchangeRates["disclaimer"]? **String**

10.What type is exchangeRates["rates"]? **[String:Double]**

11.Write a statement to typecast (using as!) exchangeRates["rates"] as the type you identified in step 10: store it in a variable, **allRates**

1. **let allRates = exchangeRates["rates"] as! [String:Double]**

12.From allRates, how will you get the conversion rate for, say, a EUR? or INR? Store it in a variable, euroRate or rupeeRate.

1.let euroRate = allRates["EUR"] ; let rupeeRate = allRates["INR"]

2.

A Solution

```
let openExchangeRatesAPI = ""

    var openExchangeRatesURL:String = ""

    // called to start the temperature fetching process
    func fetchExchangeRates() -> Void {
        openExchangeRatesURL = String("https://openexchangerates.org/api/latest.json?app_id=\(openExchangeRatesAPI)")
        print(openExchangeRatesURL)
        let urlSession = URLSession.shared
        let url = URL(string: openExchangeRatesURL)
        urlSession.dataTask(with: url!, completionHandler: displayRates).resume()
    }

func displayRates(data:Data?, urlResponse:URLResponse?, error:Error?)->Void {
        var exchangeRates:[String:Any]!
        do {
            try exchangeRates = JSONSerialization.jsonObject(with: data!, options: .allowFragments)  as!  [String:Any]
            if exchangeRates != nil {
                let allRates = exchangeRates["rates"] as! [String:Double]
                let euroRate = allRates["EUR"]
            }
         } catch {
            print(error)
        }

    }
```

References

- https://nssdc.gsfc.nasa.gov/planetary/factsheet/

- https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/URLLoadingSystem/URLLoadingSystem.html#//apple_ref/doc/uid/10000165i

- https://developer.apple.com/documentation/foundation/archives_and_serialization/using_json_with_custom_types

- https://jsonformatter.curiousconcept.com

- http://benscheirman.com/2017/06/ultimate-guide-to-json-parsing-with-swift-4/ [Very nice examples]

- https://www.themoviedb .org