Sun*

GCD, URLSession, JSON

Outline

1. Grand Central Dispatch (GCD)
2. URLSession
3. JSON

Sun*

1. Grand Central Dispatch

1. General
2. DispatchQueue
3. DispatchGroup
4. DispatchSemaphore

1.1 General

- ❖ Grand Central Dispatch (GCD) is a low-level API for managing concurrent operations. It can help you improve your app’s responsiveness by deferring computationally expensive tasks to the background. It’s an easier concurrency model to work with than locks and threads.

- ❖ GCD contains language features, runtime libraries, and system enhancements that provide systemic, comprehensive improvements to the support for concurrent code execution on multicore hardware in macOS, iOS, watchOS, and tvOS.

Sun*

1.2 DispatchQueue

- ❖ Dispatch queues are FIFO queues to which your application can submit tasks in the form of block objects.
- ❖ Dispatch queues execute tasks either serially or concurrently. Work submitted to dispatch queues executes on a pool of threads managed by the system.
- ❖ Except for the dispatch queue representing your app's main thread, the system makes no guarantees about which thread it uses to execute a task.
- ❖ You can schedule work items synchronously or asynchronously.

1.2 DispatchQueue (cont)

- ❖ There are two types of dispatch queues:
  - ➔ **Serial queues** can execute one task at a time, these queues can be utilized to synchronize access to a specific resource.
  - ➔ **Concurrent queues** on the other hand can execute one or more tasks parallel in the same time.

1.2 DispatchQueue (cont)

- Main, global and custom queues:

  - **The main queue** is a **serial** one, every task on the main queue runs on the main thread.

  - **Global queues** are system provided **concurrent** queues shared through the operating system. There are exactly four of them organized by high, default, low priority plus an IO throttled background queue.

  - **Custom queues** can be created by the user. Custom concurrent queues always mapped into one of the global queues by specifying a **Quality of Service** property (QoS).

Sun*

1.2 DispatchQueue (cont)

- System provided queues:
  - Serial main queue
  - Concurrent global queues
  - high priority global queue
  - default priority global queue
  - low priority global queue
  - global background queue (io throttled)

1.2 DispatchQueue (cont)

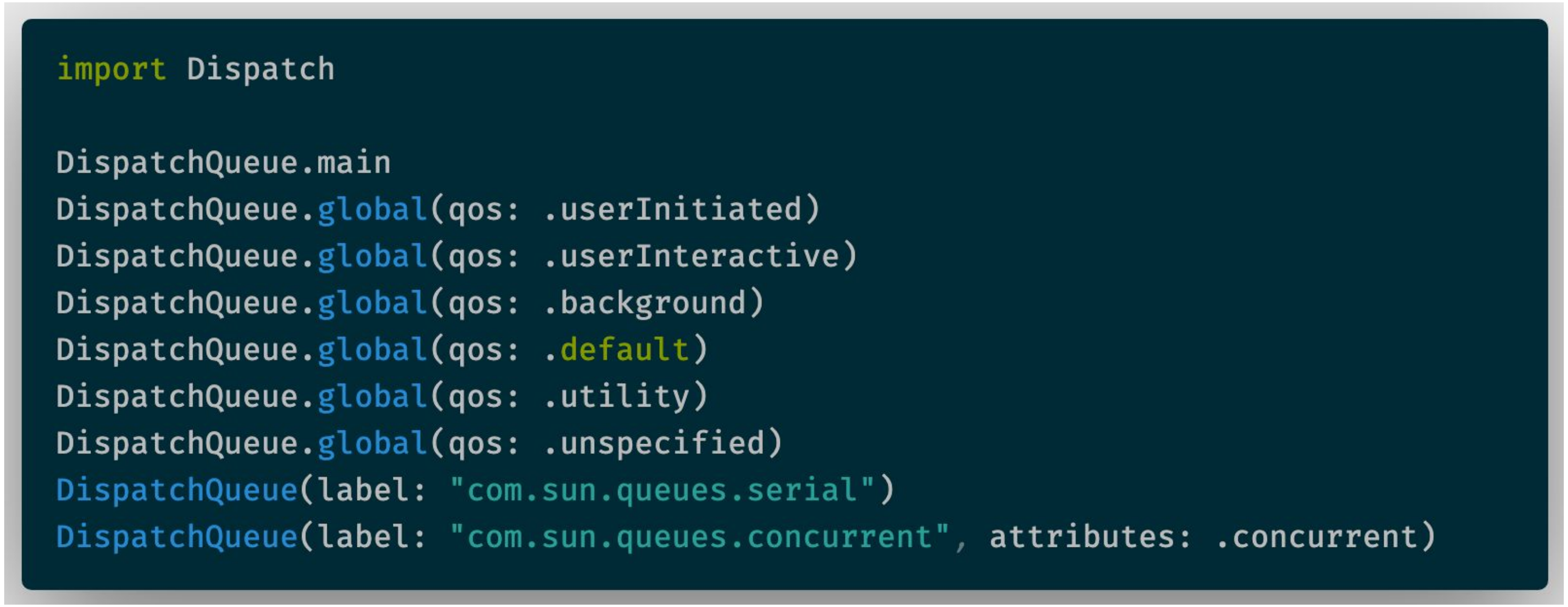
- ❖ Custom queues by quality of service:

  - ➔ userInteractive (UI updates) -> serial main queue
  - ➔ userInitiated (async UI related tasks) -> high priority global queue
  - ➔ default -> default priority global queue
  - ➔ utility -> low priority global queue
  - ➔ background -> global background queue
  - ➔ unspecified (lowest) -> low priority global queue

Sun*

1.2 DispatchQueue (cont)

❖ How to get a queue:

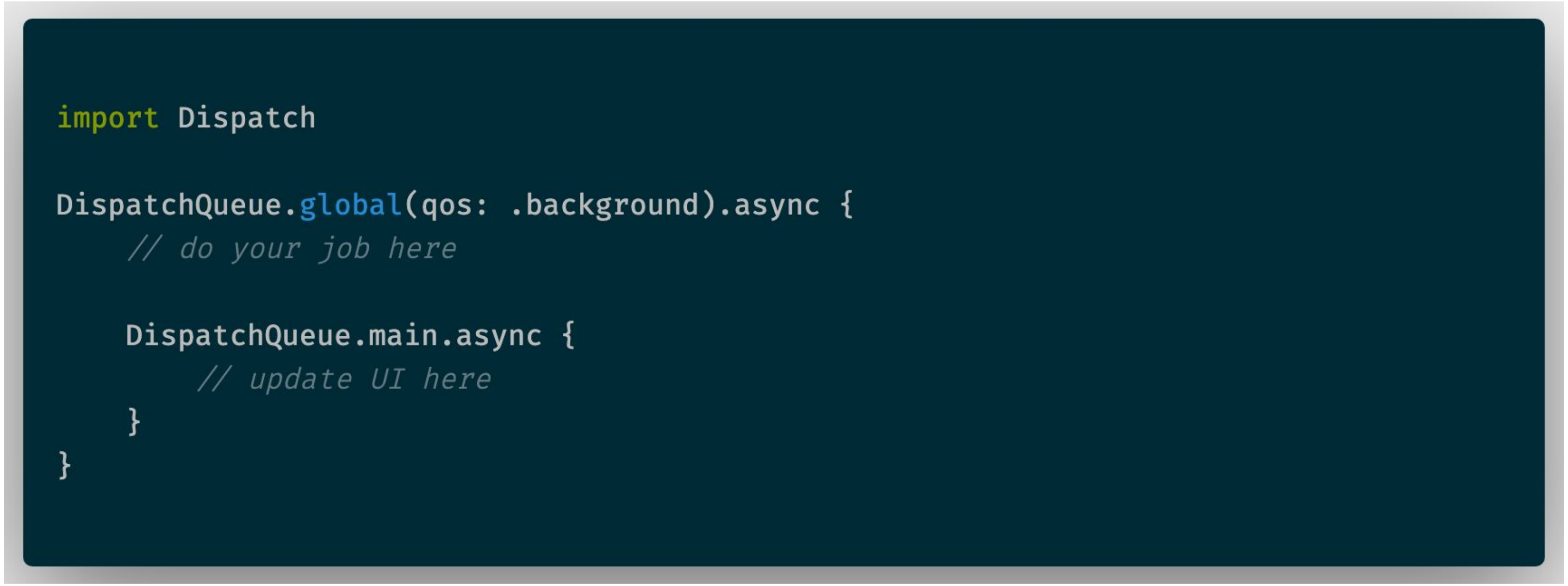


```swift
import Dispatch

DispatchQueue.main
DispatchQueue.global(qos: .userInitiated)
DispatchQueue.global(qos: .userInteractive)
DispatchQueue.global(qos: .background)
DispatchQueue.global(qos: .default)
DispatchQueue.global(qos: .utility)
DispatchQueue.global(qos: .unspecified)
DispatchQueue(label: "com.sun.queues.serial")
DispatchQueue(label: "com.sun.queues.concurrent", attributes: .concurrent)
```

1.2 DispatchQueue (cont)

❖ Executing a task on a background queue and updating the UI on the main queue after the task finished:



```
import Dispatch

DispatchQueue.global(qos: .background).async {
    // do your job here

    DispatchQueue.main.async {
        // update UI here
    }
}
```

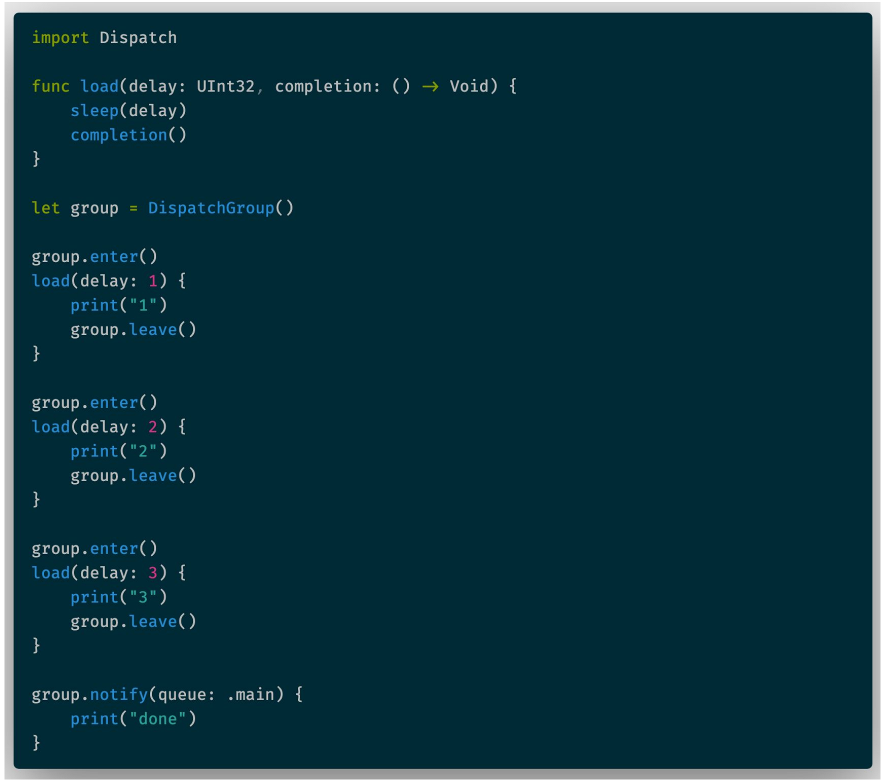
1.3 DispatchGroup

❖ Groups allow you to aggregate a set of tasks and synchronize behaviors on the group. You attach multiple work items to a group and schedule them for asynchronous execution on the same queue or different queues. When all work items finish executing, the group executes its completion handler. You can also wait synchronously for all tasks in the group to finish executing.

❖ Using dispatch groups also gives us a big advantage in that our tasks can run concurrently, in separate queues. That enables us to start off simple, and then easily add concurrency later if needed, without having to rewrite any of our tasks. All we have to do is make balanced calls to `enter()` and `leave()` on a dispatch group to have it synchronize our tasks.

1.3 DispatchGroup (cont)

❖ Example:



```swift
import Dispatch

func load(delay: UInt32, completion: () → Void) {
    sleep(delay)
    completion()
}

let group = DispatchGroup()

group.enter()
load(delay: 1) {
    print("1")
    group.leave()
}

group.enter()
load(delay: 2) {
    print("2")
    group.leave()
}

group.enter()
load(delay: 3) {
    print("3")
    group.leave()
}

group.notify(queue: .main) {
    print("done")
}
```

Sun*

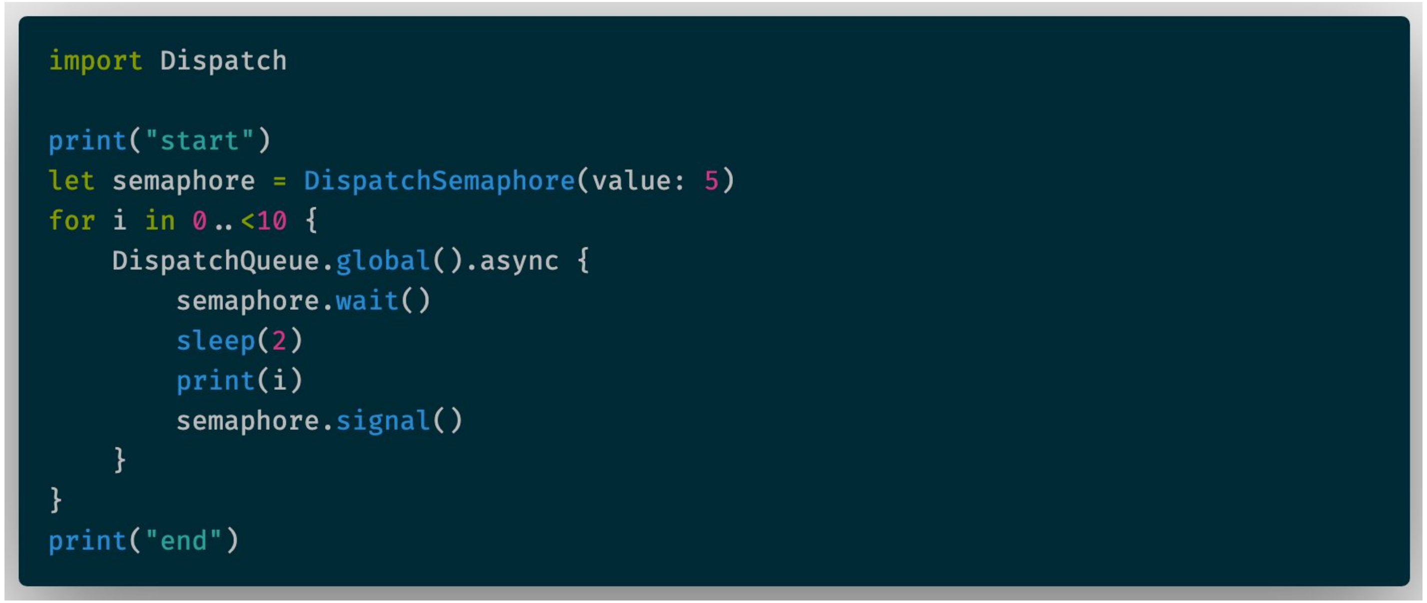
1.4 DispatchSemaphores

- ❖ A dispatch semaphore is an efficient implementation of a traditional counting semaphore. Dispatch semaphores call down to the kernel only when the calling thread needs to be blocked. If the calling semaphore does not need to block, no kernel call is made.

- ❖ You increment a semaphore count by calling the `signal()` method, and decrement a semaphore count by calling `wait()` or one of its variants that specifies a timeout.

1.4 DispatchSemaphores (cont)

❖ Example:



```
import Dispatch

print("start")
let semaphore = DispatchSemaphore(value: 5)
for i in 0..<10 {
    DispatchQueue.global().async {
        semaphore.wait()
        sleep(2)
        print(i)
        semaphore.signal()
    }
}
print("end")
```

2. URLSession

1. General
2. Types of URL Sessions
3. Types of URL Session Tasks
4. Using a Session Delegate
5. Asynchronicity and URL Sessions
6. Example

Sun*

2.1 General

❖ The `URLSession` class and related classes provide an API for downloading data from and uploading data to endpoints indicated by URLs.

❖ Your app can also use this API to perform background downloads when your app isn’t running or, in iOS, while your app is suspended.

❖ You can use the related `URLSessionDelegate` and `URLSessionTaskDelegate` to support authentication and receive events like redirection and task completion.

Sun*

2.2 Types of URL Sessions

- ❖ The tasks within a given URL session share a common session configuration object, which defines connection behavior, like the maximum number of simultaneous connections to make to a single host, whether connections can use the cellular network, and so on.

- ❖ URLSession has a singleton `shared` session (which doesn't have a configuration object) for basic requests. It's not as customizable as sessions you create, but it serves as a good starting point if you have very limited requirements.

Sun*

2.2 Types of URL Sessions (cont)

❖ For other kinds of sessions, you create a URLSession with one of three kinds of configurations:

- ➔ A default session behaves much like the shared session, but lets you configure it. You can also assign a delegate to the default session to obtain data incrementally.
- ➔ Ephemeral sessions are similar to shared sessions, but don’t write caches, cookies, or credentials to disk.
- ➔ Background sessions let you perform uploads and downloads of content in the background while your app isn't running.

Sun*

2.3 Types of URL Session Tasks

- Within a session, you create tasks that optionally upload data to a server and then retrieve data from the server either as a file on disk or as one or more `NSData` objects in memory. The `URLSession` API provides four types of tasks:
  - Data tasks send and receive data using `NSData` objects. Data tasks are intended for short, often interactive requests to a server.
  - Upload tasks are similar to data tasks, but they also send data (often in the form of a file), and support background uploads while the app isn't running.
  - Download tasks retrieve data in the form of a file, and support background downloads and uploads while the app isn't running.
  - WebSocket tasks exchange messages over TCP and TLS

2.4 Using a Session Delegate

❖ Tasks in a session also share a common delegate object. You implement this delegate to provide and obtain information when various events occur, including when:

  ➔ Authentication fails.

  ➔ Data arrives from the server.

  ➔ Data becomes available for caching.

❖ If you don’t need the features provided by a delegate, you can use this API without providing one by passing **`nil`** when you create a session.

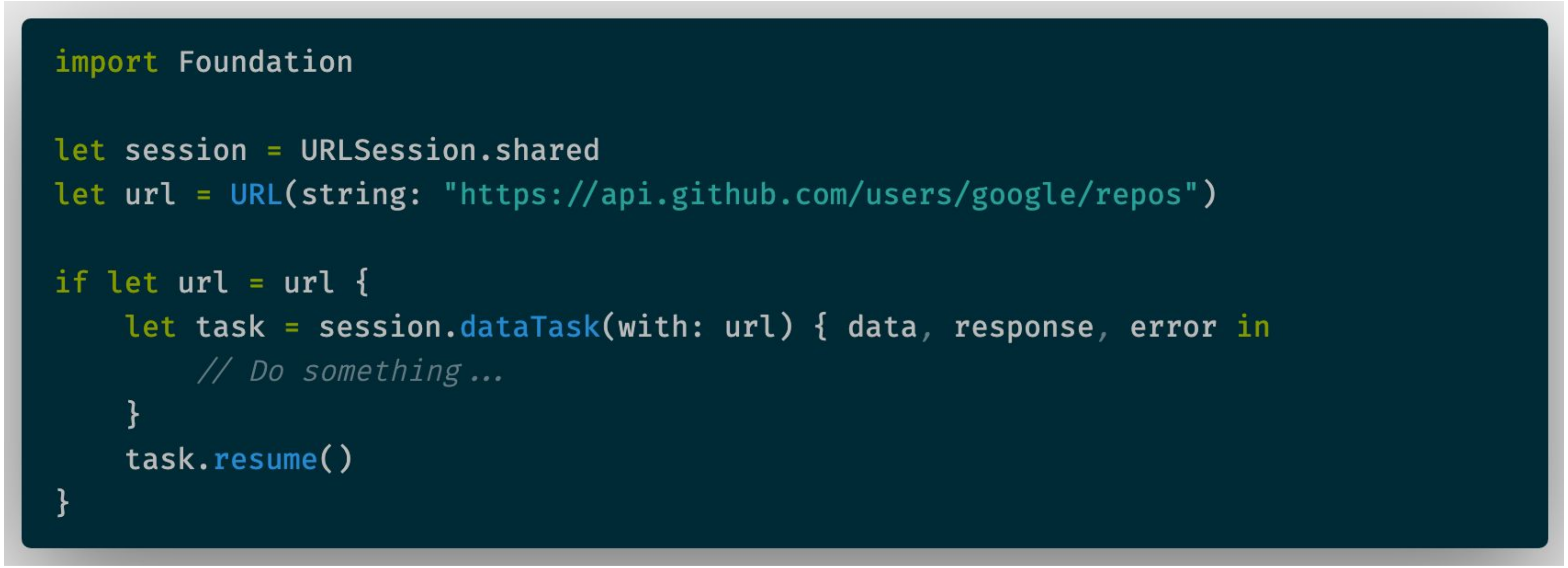
Sun*

2.5 Asynchronicity and URL Sessions

- Like most networking APIs, the `URLSession` API is highly asynchronous. It returns data to your app in one of two ways, depending on the methods you call:
  - By calling a completion handler block when a transfer finishes successfully or with an error.
  - By calling methods on the session’s delegate as data arrives and when the transfer is complete.
- In addition to delivering this information to delegates, the `URLSession` provides status and progress properties that you can query if you need to make programmatic decisions based on the current state of the task (with the caveat that its state can change at any time).

Sun*

2.6 Example



```swift
import Foundation

let session = URLSession.shared
let url = URL(string: "https://api.github.com/users/google/repos")

if let url = url {
    let task = session.dataTask(with: url) { data, response, error in
        // Do something...
    }
    task.resume()
}
```

Sun*

3. JSON

1. General
2. Serialize/Deserialize JSON with `JSONSerialization`
3. Serialize/Deserialize JSON with `ObjectMapper`

Sun*

3.1 General

❖ JSON (JavaScript Object Notation) is a lightweight data-interchange format.

➔ It is easy for humans to read and write.
➔ It is easy for machines to parse and generate.

❖ JSON is built on two structures:

➔ A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
➔ An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

3.1 General (cont)

- JSON example:

```
{
  "someKey": 42.0,
  "anotherKey": {
    "someNestedKey": true
  }
}
```

3.2 Serialize/Deserialize JSON with JSONSerialization

❖ You use the `JSONSerialization` class to convert JSON to Foundation objects and convert Foundation objects to JSON.

❖ A Foundation object that may be converted to JSON must have the following properties:

➔ The top level object is an `NSArray` or `NSDictionary`.

➔ All objects are instances of `NSString`, `NSNumber`, `NSArray`, `NSDictionary`, or `NSNull`.

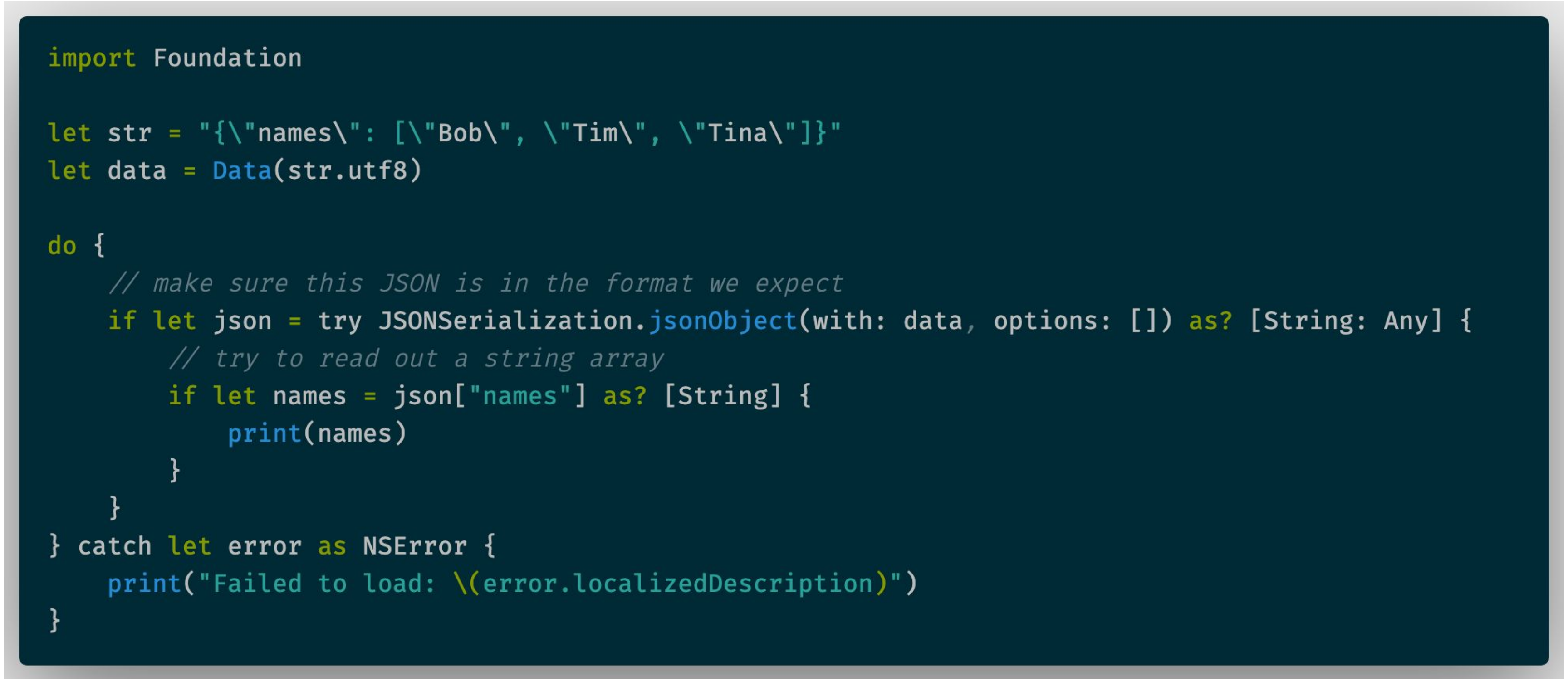
➔ All dictionary keys are instances of NSString.

➔ Numbers are not NaN or infinity.

❖ Other rules may apply. Calling `isValidJSONObject(_:)` or attempting a conversion are the definitive ways to tell if a given object can be converted to JSON data.

Sun*

3.2 Serialize/Deserialize JSON with JSONSerialization



```swift
import Foundation

let str = "{\"names\": [\"Bob\", \"Tim\", \"Tina\"]}"
let data = Data(str.utf8)

do {
    // make sure this JSON is in the format we expect
    if let json = try JSONSerialization.jsonObject(with: data, options: []) as? [String: Any] {
        // try to read out a string array
        if let names = json["names"] as? [String] {
            print(names)
        }
    }
} catch let error as NSError {
    print("Failed to load: \(error.localizedDescription)")
}
```

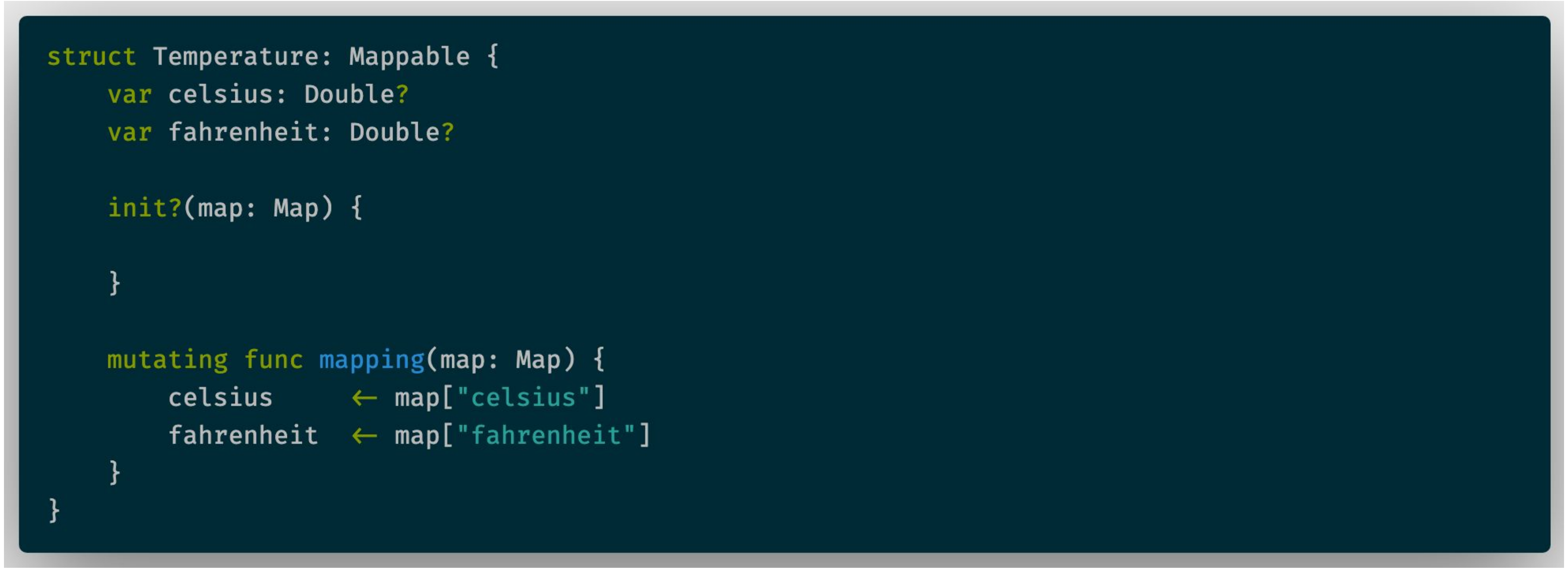
Sun*

3.3 Serialize/Deserialize JSON with ObjectMapper

- ❖ ObjectMapper is a framework written in Swift that makes it easy for you to convert your model objects (classes and structs) to and from JSON.
- ❖ Features:
  - ➔ Mapping JSON to objects
  - ➔ Mapping objects to JSON
  - ➔ Nested Objects (stand alone, in arrays or in dictionaries)
  - ➔ Custom transformations during mapping
  - ➔ Struct support
  - ➔ Immutable support
- ❖ To support mapping, a class or struct just needs to implement the Mappable protocol which includes the following functions:
  - ➔ `init?(map: Map)`
  - ➔ `mutating func mapping(map: Map)`

3.3 Serialize/Deserialize JSON with ObjectMapper

❖ Example:



```swift
struct Temperature: Mappable {
    var celsius: Double?
    var fahrenheit: Double?

    init?(map: Map) {

    }

    mutating func mapping(map: Map) {
        celsius      ← map["celsius"]
        fahrenheit   ← map["fahrenheit"]
    }
}
```

Question & Answer?

Sun*