

Networking and Error Handling

Lecture 8

CIS 1951

Last time, in CIS 1951...

Sensors

- Delegate Pattern
- Observer Pattern
- Core Motion
- Core Location
- **Questions? Comments? Feedback?**

CIS 1951 as a whole

Lectures 1-6: The Basics

Lectures 7-10: Technologies

Lectures 11-13: Beyond Development

HTTP Requests

URLSession | Apple Developer

developer.apple.com/documentation/foundation/urlsession

YouTubeWolfram AlphaSymbolabDesmos

All Bookmarks

Developer

NewsDiscoverDesignDevelopDistributeSupportAccount

Foundation

Documentation / URLSession

Language: Swift API Changes: Show

URLSession

Using the shared session

class var shared: URLSe...

Creating a session

init(configuration: URLSe...

init(configuration: URLSe...

URLSessionConfiguration

var configuration: URLSe...

Working with a delegate

var delegate: (any URLSe...

URLSessionDelegate

URLSessionTaskDelegate

var delegateQueue: Oper...

Performing asynchronous t...

Filter

/

Class

URLSession

An object that coordinates a group of related, network data transfer tasks.

iOS 7.0+ iPadOS 7.0+ macOS 10.9+ Mac Catalyst 13.1+ tvOS 9.0+ watchOS 2.0+

visionOS 1.0+

class URLSession : NSObject

Overview

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 `URLSession`

1 Make a URLRequest

```
func makeNetworkRequest() {  
    let url = URL(string: "https://demo.com")!  
    var request = URLRequest(url: url)  
    // ...  
}
```

2 Use URLSession.shared to make request

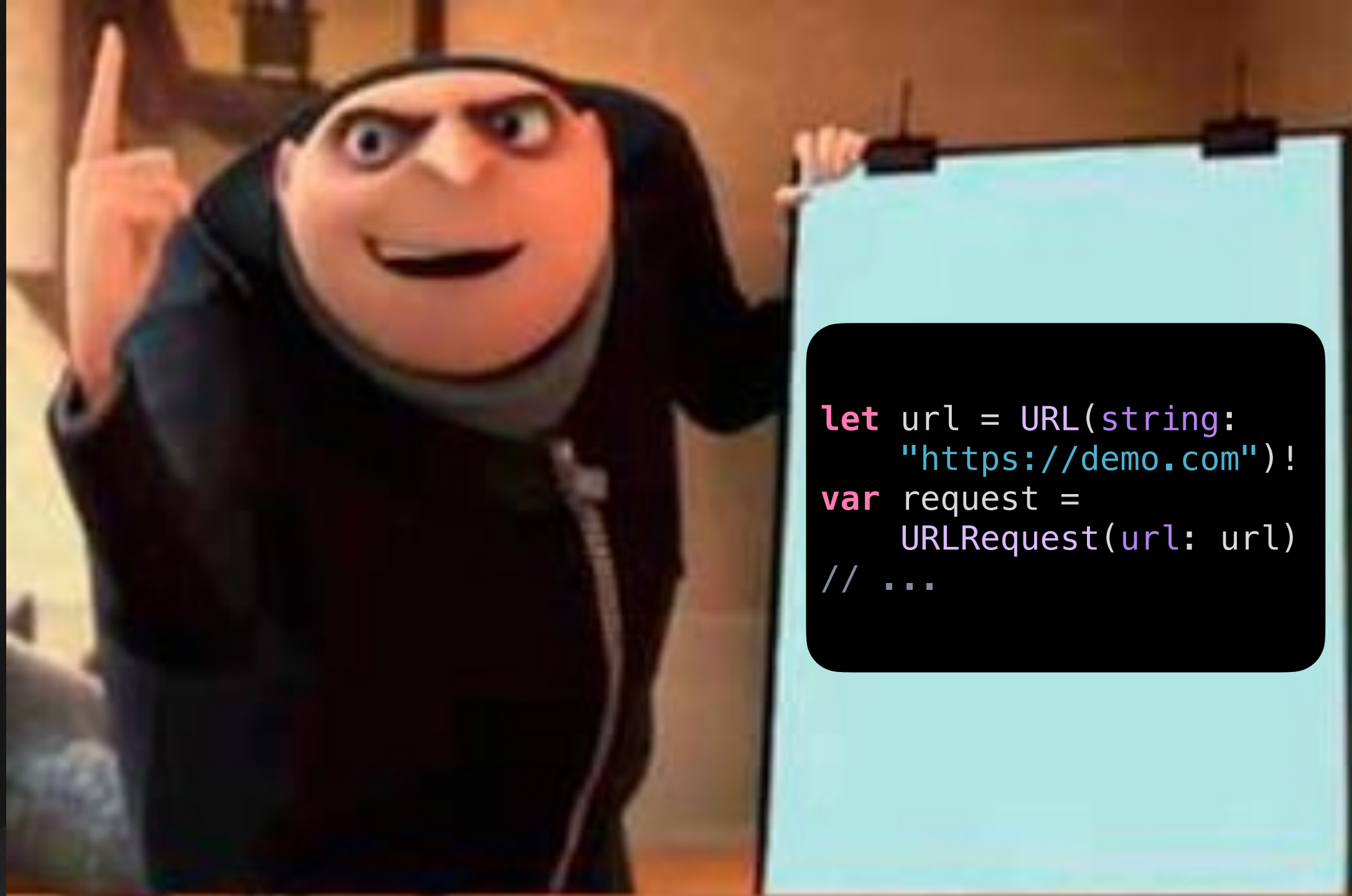
```
func makeNetworkRequest() async throws {  
    let url = URL(string: "https://demo.com")!  
    var request = URLRequest(url: url)  
  
    let (data, response) = try await URLSession.shared.data(for: request)  
    // ...  
}
```

3 Handle errors

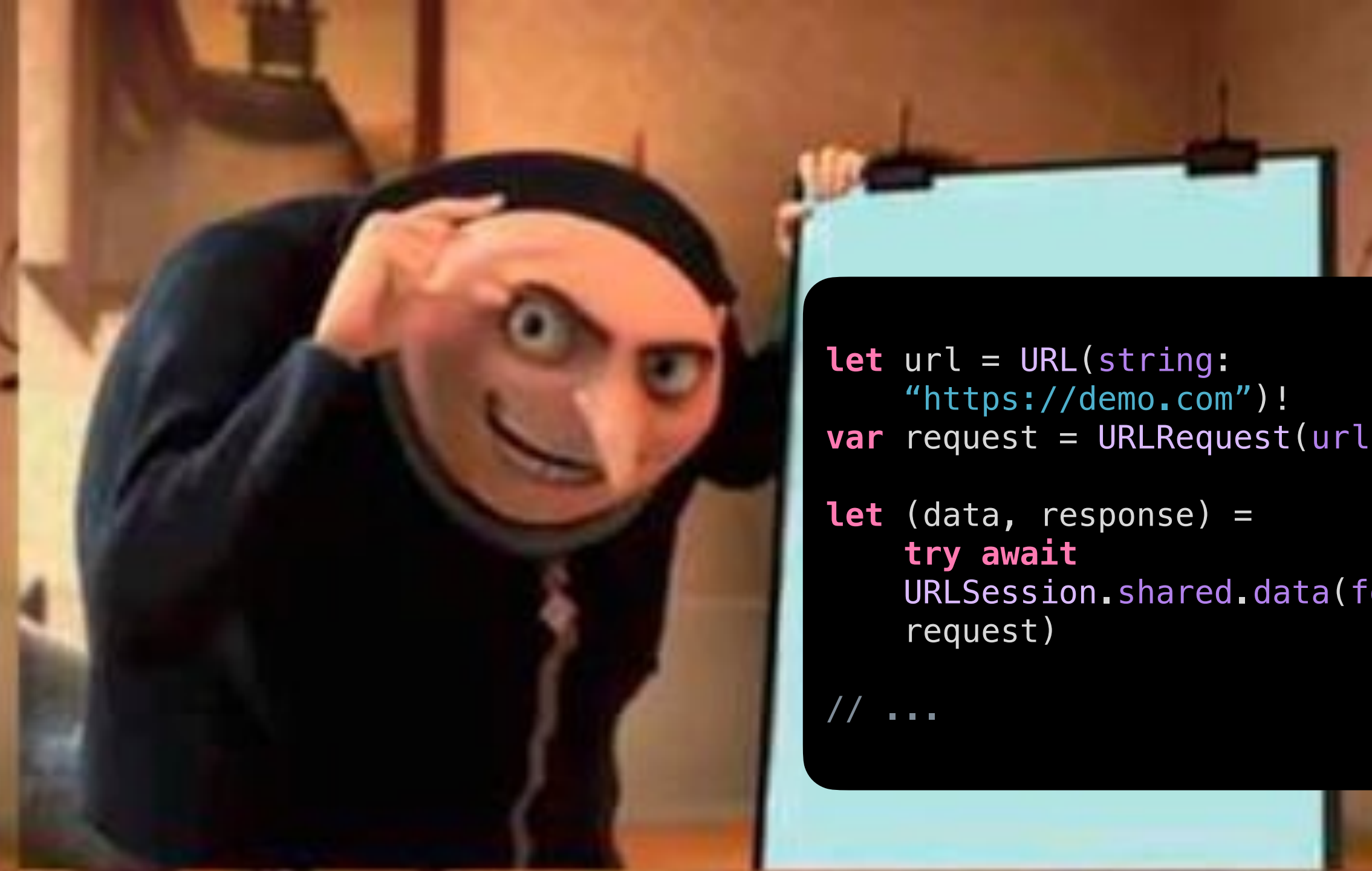
```
func makeNetworkRequest() async throws {  
    let url = URL(string: "https://demo.com")!  
    var request = URLRequest(url: url)  
  
    let (data, response) = try await URLSession.shared.data(for: request)  
  
    guard let httpResponse = response as? HTTPURLResponse,  
          httpResponse.statusCode >= 200, httpResponse.statusCode <= 299 else {  
        throw NetworkError.networkError  
    }  
    // ...  
}
```

3 Do something with the response data!

```
func makeNetworkRequest() async throws {  
    let url = URL(string: "https://demo.com")!  
    var request = URLRequest(url: url)  
  
    let (data, response) = try await URLSession.shared.data(for: request)  
  
    guard let httpResponse = response as? HTTPURLResponse,  
          httpResponse.statusCode >= 200, httpResponse.statusCode <= 299 else {  
        throw NetworkError.networkError  
    }  
  
    print(data)  
}
```



```
let url = URL(string:
    "https://demo.com")!
var request =
    URLRequest(url: url)
// ...
```



```
let url = URL(string:
    "https://demo.com")!
var request = URLRequest(url: url)

let (data, response) =
    try await
        URLSession.shared.data(for:
            request)

// ...
```



```
print(data)
```

```
21 bytes
```



```
print(data)
```

```
21 bytes
```

Returned data has type Data!

- This is NOT a string
- Basically just a list of bytes

To see string **representation**:

```
String(data: data, encoding: .utf8)
```

There are a LOT of ways to use URLSession

...that we won't have time to cover

Foundation

Performing asynchronous transfers

 func bytes(for: URLRequest, delegate: (any URLSessionTaskDelegate)?) -> (URLSession.AsyncBytes, URLResponse)

 func bytes(from: URL, delegate: (any URLSessionTaskDelegate)?) -> (URLSession.AsyncBytes, URLResponse)

>  URLSession.AsyncBytes

 func data(for: URLRequest, delegate: (any URLSessionTaskDelegate)?) -> (Data, URLResponse)

 func data(from: URL, delegate: (any URLSessionTaskDelegate)?) -> (Data, URLResponse)

 func download(for: URLRequest, delegate: (any URLSessionTaskDelegate)?) -> (URL, URLResponse)

 func download(from: URL, delegate: (any URLSessionTaskDelegate)?) -> (URL, URLResponse)

 func download(resumeFrom: Data, delegate: (any URLSessionTaskDelegate)?) -> (URL, URLResponse)

 func upload(for: URLRequest, from: Data, delegate: (any URLSessionTaskDelegate)?) -> (Data, URLResponse)

 func upload(for: URLRequest, fromFile: URL, delegate: (any URLSessionTaskDelegate)?) -> (Data, URLResponse)

>  URLSessionTaskDelegate

Adding data tasks to a session

 func dataTask(with: URL) -> URLSessionDataTask

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

 func dataTask(with: URLRequest) -> URLSessionDataTask

 func dataTask(with: URLRequest, completionHandler: (Data?, URLResponse?, (any Error)?) -> Void) -> URLSessionDataTask

>  URLSessionDataTask

>  URLSessionDataDelegate

Adding download tasks to a session

 func downloadTask(with: URL) -> URLSessionDownloadTask

 func downloadTask(with: URL, completionHandler: (URL?, URLResponse?, (any Error)?) -> Void) -> URLSessionDownloadTask

 func downloadTask(with: URLRequest) -> URLSessionDownloadTask

Session Management

`URLSession.shared` keeps track of cookies and other session related information

Use as needed & check out documentation!

Error Handling

try, try!, and try?

```
let (data, response) = try await URLSession.shared.data(for: request)
```

Error thrown is propagated

```
let (data, response) = try! await URLSession.shared.data(for: request)
```

Will crash if error is thrown

```
let (data, response) = try? await URLSession.shared.data(for: request)
```

Will return optional of result, nil if error is thrown

Do-Catch

```
do {  
    let (data, response) = try await URLSession.shared.data(for: request)  
} catch let error {  
    print(error)  
}
```

Do-Catch

```
do {  
    let (data, response) = try await URLSession.shared.data(for: request)  
} catch let error {  
    print(error)  
}
```

What does this do?

```
if let (data, response) = try? await URLSession.shared.data(for: request) {  
    // ...  
}
```

Doing Useful things with Data: Encodable & Decodable

Detour: JSON

```
{
  "orders": [
    {
      "orderno": "748745375",
      "date": "June 30, 2088 1:54:23 AM",
      "trackingno": "TN0039291",
      "custid": "11045",
      "customers": [
        {
          "custid": "11045",
          "fname": "Sue",
          "lname": "Hatfield",
          "address": "1409 Silver Street",
          "city": "Ashland",
          "state": "NE",
          "zip": "68003"
        }
      ]
    }
  ]
}
```

Receiving Data in requests

1 Make struct matching expected JSON response

```
struct Post: Identifiable, Hashable, Decodable {  
    let id: UUID  
    let author: String  
    let content: String  
    let createdAt: Date  
}
```

2 Use JSONDecoder to decode data

```
let decoder = JSONDecoder()  
let post = try decoder.decode(Post.self, from: data)
```

or

```
var decoder = JSONDecoder()  
decoder.dateDecodingStrategy = .iso8601  
decoder.keyDecodingStrategy = .convertFromSnakeCase  
let post = try decoder.decode(Post.self, from: data)
```

3

Thats it!

3

Thats it!

But what if we want specialized decoding instructions...???

Specialized Decoding

```
struct Post: Identifiable, Hashable, Decodable {
    let id: UUID
    let author: String
    let content: String?
    let createdAt: Date

    enum CodingKeys: String, CodingKey {
        case id
        case author = "creator"
        case content
    }

    init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)

        self.id = try container.decode(UUID.self, forKey: .id)
        self.author = try container.decode(String.self, forKey: .author)
        self.content = try container.decodeIfPresent(String.self, forKey: .content)
        self.createdAt = Date()
    }
}
```

Sending Data in requests

1 Make struct conform to Encodable

```
struct Post: Identifiable, Hashable, Encodable {  
    let id: UUID  
    let author: String  
    let content: String  
    let createdAt: Date  
}
```

2 Use JSONEncoder to encode data

```
let post: Post = ...  
let encoder = JSONEncoder()  
let encodedData = try encoder.encode(post)
```

or

```
let post: Post = ...  
let encoder = JSONEncoder()  
encoder.dateEncodingStrategy = .iso8601  
encoder.keyEncodingStrategy = .convertToSnakeCase  
let encodedData = try encoder.encode(post)
```

What is the type of encodedData?

```
let encodedData = try encoder.encode(post)
```

What is the type of encodedData?

```
let encodedData = try encoder.encode(post)
```

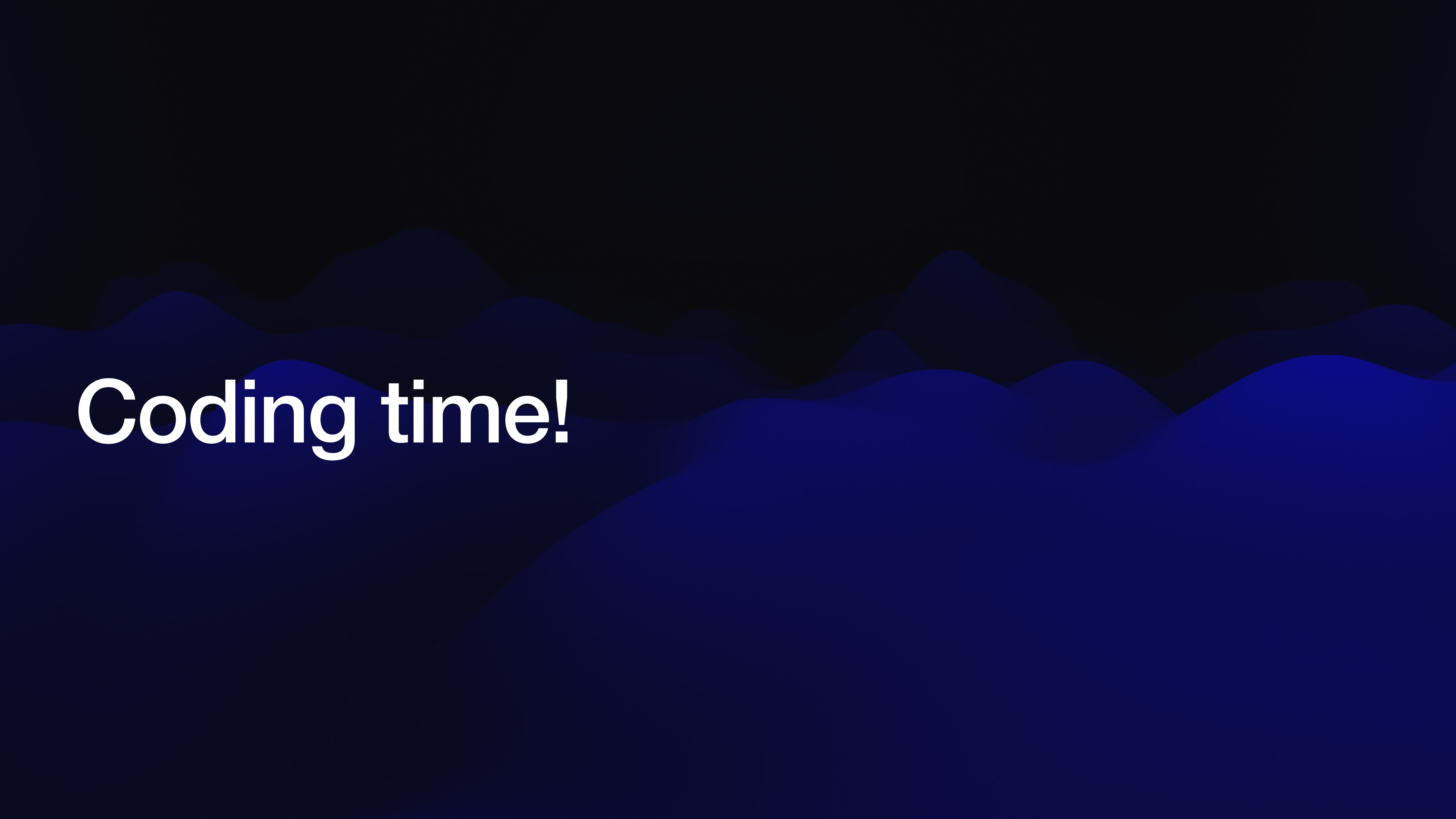
Hint: JSONDecoder decodes from type Data

Sending JSON data in network request

```
var request = URLRequest(url: url)
request.httpMethod = "POST"
request.addValue("application/json", forHTTPHeaderField: "Content-Type")
request.httpBody = try JSONEncoder().encode(post)
```

Codable = Decodable & Encodable

```
struct Post: Identifiable, Hashable, Codable {  
    let id: UUID  
    let author: String  
    let content: String  
    let createdAt: Date  
}
```

The background features a series of overlapping, wavy shapes in various shades of dark blue and black, creating a layered, mountain-like effect. The text "Coding time!" is positioned on the left side, overlaid on these shapes.

Coding time!

<https://github.com/cis1951/lec8-code>