Introduction to iOS development with Swift

Lesson 7

Adrien Humilière
Trainline

adhumi+dant@gmail.com

MAY 17

- → Closures
- → Extensions
- → HTTP and URL Session
- → JSON Serialization
- → Concurrency

Closures

Closures

```
(firstTrack: Track, secondTrack: Track) -> Bool in
 return firstTrack.trackNumber < secondTrack.trackNumber
```

```
let sortedTracks = tracks.sorted (    )
```

Syntax

```
func sum(numbers: [Int]) -> Int {
  // Code that adds together the numbers array
  return total
}
```

```
let sumClosure = { (numbers: [Int]) -> Int in
  // Code that adds together the numbers array
  return total
}
```

```
let printClosure = { () -> Void in
  print("This closure does not take any parameters and does not
return a value.")
}

let printClosure = { (string: String) -> Void in
  print(string)
}

let randomNumberClosure = { () -> Int in
  // Code that returns a random number
}

let randomNumberClosure = { (minValue: Int, maxValue: Int) -> Int in
  // Code that returns a random number between `minValue` and
`maxValue`
}
```

Passing closures as arguments

```
let sortedTracks = tracks.sorted { (firstTrack: Track,
secondTrack: Track) -> Bool in
  return firstTrack.trackNumber < secondTrack.trackNumber
}
```

```
let sortedTracks = tracks.sorted { (firstTrack: Track,
secondTrack: Track) -> Bool in
  return firstTrack.starRating < secondTrack.starRating
}
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { (firstTrack: Track,
secondTrack: Track) -> Bool in
  return firstTrack.starRating < secondTrack.starRating
}
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { (firstTrack, secondTrack) ->
Bool in
  return firstTrack.starRating < secondTrack.starRating
}
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { (firstTrack, secondTrack) in
  return firstTrack.starRating < secondTrack.starRating
}
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { return $0.starRating <
$1.starRating }
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { $0.starRating <
$1.starRating }
```

Collection functions using closures

- Map
- Filter
- Reduce

Collection functions using closures

```swift
// Initial array
let firstNames = ["Johnny", "Nellie", "Aaron", "Rachel"]

// Creates an empty array that will be used
// to store the full names
var fullNames: [String] = []

for name in firstNames {
    let fullName = name + " Smith"
    fullNames.append(fullName)
}
```

Collection functions using closures

```
// Initial array
let firstNames = ["Johnny", "Nellie", "Aaron", "Rachel"]

// Creates a new array of full names by adding "Smith"
// to each first name
let fullNames = firstNames.map { (name) -> String in
    return name + " Smith"
}
```

Collection functions using closures

```
// Initial array
let firstNames = ["Johnny", "Nellie", "Aaron", "Rachel"]

// Creates a new array of full names by adding "Smith"
// to each first name
let fullNames = firstNames.map{ $0 + " Smith" }
```

Collection functions using closures

```
let numbers = [4, 8, 15, 16, 23, 42]

var numbersLessThan20: [Int] = []

for number in numbers {
    if number < 20 {
        numbersLessThan20.append(number)
    }
}
```

Collection functions using closures

```
let numbers = [4, 8, 15, 16, 23, 42]

let numbersLessThan20 = numbers.filter { (number) -> Bool in
    return number < 20
}
```

Collection functions using closures

```
let numbers = [4, 8, 15, 16, 23, 42]

let numbersLessThan20 = numbers.filter{ $0 < 20 }
```

Collection functions using closures

```
let numbers = [8, 6, 7, 5, 3, 0, 9]

var total = 0

for number in numbers {
    total = total + number
}
```

Collection functions using closures

```
let numbers = [8, 6, 7, 5, 3, 0, 9]

let total = numbers.reduce(0) { (currentTotal, newValue) ->
Int in
    return currentTotal + newValue
}
```

Collection functions using closures

```
let numbers = [8, 6, 7, 5, 3, 0, 9]

let total = numbers.reduce(0,{ $0 + $1})
```

Extensions

🚂🚃

Extensions

```
extension SomeType {
  // new functionality to add to SomeType goes here
}
```

Adding computed properties

```swift
extension UIColor {
  static var favoriteColor: UIColor {
    return UIColor(red: 0.5, green: 0.1, blue: 0.5, alpha: 1.0)
  }
}
```

Adding instance or type methods

```
extension String {
  func pluralized() -> String {
    // Complex code that takes the current value (self) and
returns the plural version
  }
}

var apple = "Apple"
var person = "Person"

print(apple.pluralized()) // Apples
print(person.pluralized()) // People
```

Organizing code

```
class Restaurant {
  let name: String

  var menuItems: [MenuItem]
  . . .
}

extension Restaurant {
  func add(menuItem: MenuItem)
  func remove(menuItem: MenuItem)
}
```

HTTP and URL Session

📡

Basics

```
https://sales.pretendco.com:80/orders/strack?
order=233282&api_key=QREPORT
```

HTTP methods

Method	Description
GET	Requests information from a server
POST	Sends information to a server
PUT	Updates information from a server
DELETE	Deletes information from a server

HTTP headers

Allows the client and the server to exchange information

→ Used for authentication

→ Sends information such as the computer or browser type to the server

→ Responds with information such as the server type and software used to handle the request

HTTP body

Includes the data sent from the client or server following the HTTP headers

- → Sends form data to the server
- → Responds with a web page content and images

Network request

```
let url = URL(string: "https://www.apple.com")!
let task = URLSession.shared.dataTask(with: url) { (data,
response, error) in
    if let data = data,
       let string = String(data: data, encoding: .utf8) {
         print(string)
    }
}
task.resume()
```

Work with an API

```
let url = URL(string: "https://api.nasa.gov/planetary/apod?
date=2005-2-22&api_key=DEMO_KEY")!

let task = URLSession.shared.dataTask(with: url) { (data,
response, error) in
    if let data = data,
        let string = String(data: data, encoding: .utf8) {

        print(string)
    }
}
task.resume()
```

URL Components

```swift
extension URL {
    func withQueries(_ queries: [String: String]) -> URL? {
        var components = URLComponents(url: self,
                                       resolvingAgainstBaseURL: true)
        components?.queryItems = queries.flatMap {
            URLQueryItem(name: $0.0, value: $0.1)
        }
        return components?.url
    }
}
```

```swift
let baseURL = URL(string: "https://api.nasa.gov/planetary/apod")!
let query: [String: String] = [
    "api_key": "DEMO_KEY",
    "date": "2011-07-13"
]
let url = baseURL.withQueries(query)!
let task = URLSession.shared.dataTask(with: url) { (data,
response, error) in
    if let data = data,
        let string = String(data: data, encoding: .utf8) {
        print(string)

    } }
task.resume()
```

Decoding JSON

JSON

```
{
    "name": "Daren Estrada",
    "favorite_movie": {
        "title": "Finding Dory",
        "release_year": "2016"
    }
}
```

An open standard format that uses human readable text to transmit objects

→ Each object consists of attribute-value pairs

Used primarily to transmit data between a server and applications

Language-independent data format

JSON basics

```
{
    "name": "Daren Estrada",
    "favorite_movies": [
        {
            "title": "Finding Dory",
            "release_year": 2016
        },
        {
            "title": "Inside Out",
            "release_year": 2015
        }
    ]
}
```

JSON data to Swift types

```
let task = URLSession.shared.dataTask(with: url) { (data,
response, error) in
    if let data = data,
        let jsonDecoder = JSONDecoder()
        let report = try? jsonDecoder.decode([String:
String].self, from: data) {
        print(report)
    }
}

task.resume()
```

Concurrency

🔀

Concurrency

- Run multiple tasks at the same time
- Run slow or expensive tasks in the background
- Free the main thread so it responds to the UI

Synchronous and asynchronous

Synchronous

- One task completes before another begins
- Ties up the main thread (main queue)

Asynchronous

- Multiple tasks run simultaneously on multiple threads (concurrency)
- Tasks run in the background thread (background queue)
- Frees up the main thread

Grand Central Dispatch

Grand Central Dispatch

- Allows your app to execute multiple tasks concurrently on multiple threads
- Assigns tasks to "dispatch queues" and assigns priority
- Controls when your code is executed

Grand Central Dispatch

- Main queue
  - Created when an app launches
  - Highest priority
  - Used to update the UI and respond quickly to user input
- Background queues
  - Lower-priority
  - Used to run long-running operations

Dispatch Queue

Use the DispatchQueue type to create and assign tasks to different queues

For example:

→ Assign a UI task to the main dispatch queue

→ Tasks added with main.async(…) run sequentially

```
DispatchQueue.main.async {
    // Code here will be executed on the main queue
}
```

App Transport Security

- ATS improves user security and privacy
- Requires apps to use secure network connections over HTTPS

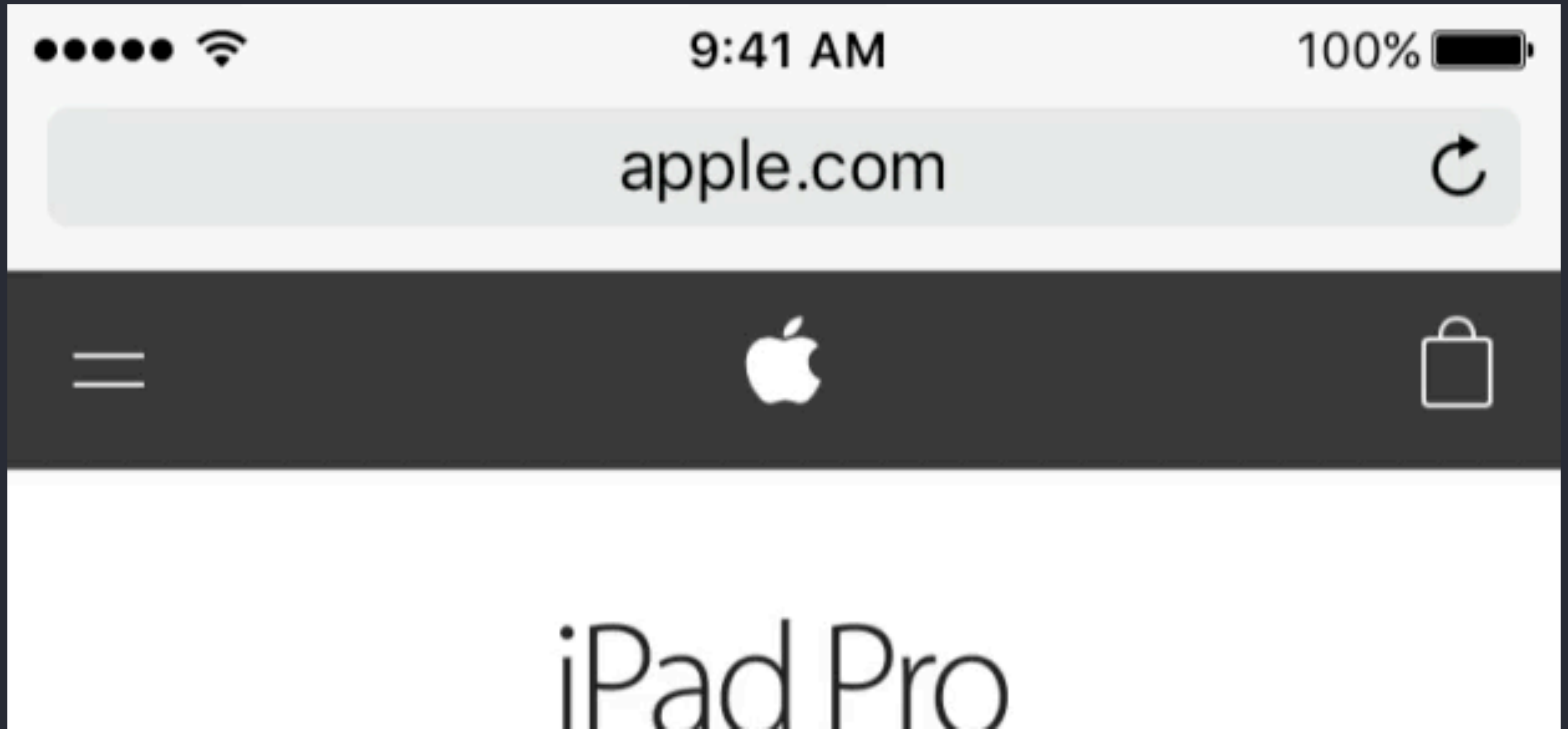
```
extension URL {
  func withHTTPS() -> URL? {
      var components = URLComponents(url: self,
resolvingAgainstBaseURL: true)
      components?.scheme = "https"

      return components?.url
  }
}
```

Network activity indicator

→ Shows that your app is executing a network request and waiting for a response

```
UIApplication.shared.isNetworkActivityIndicatorVisible = true
```



The End.