

# Introduction to iOS development with Swift

## Lesson 2



**Adrien Humilière**  
Brut.

[adhumi+dant@gmail.com](mailto:adhumi+dant@gmail.com)



- Optionals
- Type casting
- Guard
- Scope
- Closures
- Extensions
- HTTP and URL Session
- JSON Serialization
- Concurrency
- Protocols

# Optionals



# nil

```
struct Book {  
    let name: String  
    let publicationYear: Int  
}
```

```
let firstHarryPotter = Book(name: "Harry Potter and the  
Sorcerer's Stone", publicationYear: 1997)  
let secondHarryPotter = Book(name: "Harry Potter and the  
Chamber of Secrets", publicationYear: 1998)
```

```
let books = [firstHarryPotter, secondHarryPotter]
```

# nil

```
let unannouncedBook = Book(name: "Harry Potter 8",  
                             publicationYear: ???)
```

# nil

```
let unannouncedBook = Book(name: "Harry Potter 8",  
                             publicationYear: 0)
```

# nil

```
let unannouncedBook = Book(name: "Harry Potter 8",  
                             publicationYear: 2019)
```

# nil

```
let unannouncedBook = Book(name: "Harry Potter 8",  
                             publicationYear: nil)
```



Nil is not compatible with expected argument type 'Int'



```
struct Book {  
  let name: String  
  let publicationYear: Int?  
}
```

```
let firstHarryPotter = Book(name: "Harry Potter and the  
Sorcerer's Stone", publicationYear: 1997)  
let secondHarryPotter = Book(name: "Harry Potter and the  
Chamber of Secrets", publicationYear: 1998)
```

```
let books = [firstHarryPotter, secondHarryPotter]
```

```
let unannouncedBook = Book(name: "Rebels and Lions",  
publicationYear: nil)
```

# Specifying the type of an optional

```
var serverResponseCode: Int = 404
```

```
var serverResponseCode: Int = nil
```

 **'nil' requires a contextual type**

```
var serverResponseCode: Int? = 404
```

```
var serverResponseCode: Int? = nil
```

# Working with optional values

```
if publicationYear != nil {  
    let actualYear = publicationYear!  
    print(actualYear)  
}
```

```
let unwrappedYear = publicationYear!
```



error: Execution was interrupted

# Working with optional values

```
if let constantName = someOptional {  
    //constantName has been safely unwrapped for use within {}  
}
```

```
if let unwrappedPublicationYear = book.publicationYear {  
    print("The book was published in \(unwrappedPublicationYear)")  
} else {  
    print("The book does not have an official publication date.")  
}
```

# Functions and optionals

```
let string = "123"  
let possibleNumber = Int(string)
```

```
let string = "Cynthia"  
let possibleNumber = Int(string)
```

# Functions and optionals

```
func printFullName(firstName: String, middleName: String?,  
lastName: String)
```

```
func textFromURL(url: URL) -> String?
```

# Failable initializers

```
struct Toddler {  
    var birthName: String  
    var monthsOld: Int  
}
```

# Failable initializers

```
init?(birthName: String, monthsOld: Int) {  
    if monthsOld < 12 || monthsOld > 36 {  
        return nil  
    } else {  
        self.birthName = birthName  
        self.monthsOld = monthsOld  
    }  
}
```



# Failable initializers

```
let possibleToddler = Toddler(birthName: "Joanna", monthsOld: 14)
if let toddler = possibleToddler {
    print("\(toddler.birthName) is \(toddler.monthsOld) months old")
} else {
    print("The age you specified for the toddler is not between 1
and 3 yrs of age")
}
```

# Optional chaining

```
class Person {  
    var age: Int  
    var residence: Residence?  
}  
  
class Residence {  
    var address: Address?  
}
```

```
class Address {  
    var buildingNumber: String?  
    var streetName: String?  
    var apartmentNumber: String?  
}
```

# Optional chaining

```
if let theResidence = person.residence {  
    if let theAddress = theResidence.address {  
        if let theApartmentNumber = theAddress.apartmentNumber {  
            print("He/she lives in apartment number  
                \theApartmentNumber). »")  
        }  
    }  
}
```

# Implicitly Unwrapped Optionals

```
class ViewController: UIViewController {  
    @IBOutlet weak var label: UILabel!  
}
```

## **Unwraps automatically**

Should only be used when need to initialize an object without supplying the value and you'll be giving the object a value soon afterwards

# Type Casting and Inspection



# Optional chaining

```
func getClientPet() -> Animal {  
    //returns the pet  
}
```

```
let pet = getClientPet() //`pet` is of type `Animal`
```

# Optional chaining

```
if pet is Dog {  
    print("The client's pet is a dog")  
} else if pet is Cat {  
    print("The client's pet is a cat")  
} else if pet is Bird {  
    print("The client's pet is a bird")  
} else {  
    print("The client has a very exotic pet")  
}
```

# Optional chaining

```
let pets = allPets() //`pets` is of type `[Animal]`  
var dogCount = 0, catCount = 0, birdCount = 0  
for pet in pets {  
    if pet is Dog {  
        dogCount += 1  
    } else if pet is Cat {  
        catCount += 1  
    } else if pet is Bird {  
        birdCount += 1  
    }  
}  
print("Brad looks after \$(dogCount) dogs, \$(catCount) cats,  
and \$(birdCount) birds.")
```



# Type casting

```
func walk(dog: Dog) {  
    print("Walking \ (dog.name)")  
}  
  
func cleanLitterBox(cat: Cat) {. . .}  
  
func cleanCage(bird: Bird) {. . .}  
  
for pet in pets {  
    if pet is Dog {  
        walk(dog: pet) // Compiler error  
    }  
    ...  
}
```

# Type casting

```
for pet in pets {  
    if let dog = pet as? Dog {  
        walk(dog: dog)  
    } else if let cat = pet as? Cat {  
        cleanLitterBox(cat: cat)  
    } else if let bird = pet as? Bird {  
        cleanCage(bird: bird)  
    }  
}
```

# Any

```
var items: [Any] = [5, "Bill", 6.7, Dog()]
```

# Any

```
var items: [Any] = [5, "Bill", 6.7, Dog()]
let firstItem = items[0]

if firstItem is Int {
    print("The first element is an integer")
} else if firstItem is String {
    print("The first element is a string")
} else {
    print("The first element is neither an integer nor a string")
}
```

# Any

```
var items: [Any] = [5, "Bill", 6.7, Dog()]

if let firstItem = items[0] as? Int {
    print(firstItem + 4)
}
```

# Guard



```
func singHappyBirthday() {  
    if birthdayIsToday {  
        if invitedGuests > 0 {  
            if cakeCandlesLit {  
                print("Happy Birthday to you!")  
            } else {  
                print("The cake candle's haven't been lit.")  
            }  
        } else {  
            print("It's just a family party.")  
        }  
    } else {  
        print("No one has a birthday today.")  
    }  
}
```

```
func singHappyBirthday() {  
    guard birthdayIsToday else {  
        print("No one has a birthday today.")  
        return  
    }  
    guard invitedGuests > 0 else {  
        print("It's just a family party.")  
        return  
    }  
    guard cakeCandlesLit else {  
        print("The cake's candles haven't been lit.")  
        return  
    }  
    print("Happy Birthday to you!")  
}
```



# guard

```
guard condition else {  
    //false: execute some code  
}
```

```
//true: execute some code
```

# guard

```
func divide(_ number: Double, by divisor: Double) {  
    if divisor != 0.0 {  
        let result = number / divisor  
        print(result)  
    }  
}
```

```
func divide(_ number: Double, by divisor: Double) {  
    guard divisor != 0.0 else { return }  
  
    let result = number / divisor  
    print(result)  
}
```

```
func processBook(title: String?, price: Double?, pages: Int?) {  
    if let theTitle = title, let thePrice = price, let thePages =  
pages {  
        print("\(theTitle) costs $\(thePrice) and has \(thePages)  
pages.")  
    }  
}
```

```
func processBook(title: String?, price: Double?, pages: Int?){  
    guard let theTitle = title, let thePrice = price, let  
thePages = pages else { return }  
    print("\(theTitle) costs $\(thePrice) and has \(thePages)  
pages.")  
}
```

# Constant and Variable Scope



# Scope

**Global scope** — Defined outside of a function

**Local scope** — Defined within braces ({} )

```
var globalVariable = true

if globalVariable {
  let localVariable = 7
}
```

# Scope

```
var age = 55

func printMyAge() {
    print("My age: \(age)")
}

print(age)
printMyAge()
```

# Scope

```
func printBottleCount() {  
    let bottleCount = 99  
    print(bottleCount)  
}
```

```
printBottleCount()  
print(bottleCount)
```



Use of unresolved identifier 'bottleCount'

# Scope

```
func printTenNames() {  
    var name = "Richard"  
    for index in 1...10 {  
        print("\(index): \(name)")  
    }  
    print(index)  
    print(name)  
}  
  
printTenNames()
```



Use of unresolved identifier 'index'



# Variable shadowing

```
let points = 100

for index in 1...3 {
    let points = 200
    print("Loop \(index): \(points+index)")
}
print(points)
```

# Variable shadowing

```
var name: String? = "Robert"

if let name = name {
    print("My name is \(name)")
}
```

# Variable shadowing

```
func exclaim(name: String?) {  
    if let name = name {  
        print("Exclaim function was passed: \(name)")  
    }  
}
```

```
func exclaim(name: String?) {  
    guard let name = name else { return }  
    print("Exclaim function was passed: \(name)")  
}
```

# Shadowing and initializers

```
struct Person {  
    var name: String  
    var age: Int  
}
```

```
let todd = Person(name: "Todd", age: 50)  
print(todd.name)  
print(todd.age)
```

# Shadowing and initializers

```
struct Person {  
    var name: String  
    var age: Int  
  
    init(name: String, age: Int) {  
        self.name = name  
        self.age = age  
    }  
}
```

# 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

```
// 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

```
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

```
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  
    }  
}
```

```
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  
    let jsonDecoder = JSONDecoder()  
    if let data = data,  
        let report = try? jsonDecoder.decode(Rreport.self,  
from: data) {  
        print(report)  
    }  
}  
  
task.resume()
```

# JSON data to custom model objects

```
{  
  "report_date": "2018-01-20",  
  "profile_id": "136442",  
  "name": "Final Results for Q4 2017",  
  "read_count": "5"  
}
```

# JSON data to custom model objects

```
struct Report {  
    let name: String  
    let creationDate: Date  
    let profileID: String  
    let readCount: Int?  
}
```

# JSON data to custom model objects

```
struct Report: Codable {  
    let name: String  
    let creationDate: Date  
    let profileID: String  
    let readCount: Int?  
  
    enum CodingKeys: String, CodingKey {  
        case name  
        case creationDate = "report_date"  
        case profileID = "profile_id"  
        case readCount = "read_count"  
    }  
}
```

# JSON data to custom model objects

```
init(from decoder: Decoder) throws {  
    let valueContainer = try decoder.container(keyedBy:  
        CodingKeys.self)  
    self.creationDate = try valueContainer.decode(String.self,  
        forKey: CodingKeys.creationDate)  
    self.profileID = try valueContainer.decode(URL.self,  
        forKey: CodingKeys.profileID)  
    self.readCount = try? valueContainer.decode(String.self,  
        forKey: CodingKeys.readCount)  
}
```

# JSON data to Swift types

```
let task = URLSession.shared.dataTask(with: url) { (data,  
response, error) in  
    let jsonDecoder = JSONDecoder()  
    if let data = data,  
        let report = try? jsonDecoder.decode(Rreport.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  
}
```

# Protocols





# Protocols

- Defines a blueprint of methods, properties, and other requirements that suit a particular task or piece of functionality
- Swift standard library defines many protocols, including these:
  - `CustomStringConvertible`
  - `Equatable`
  - `Comparable`
  - `Codable`
- When you adopt a protocol, you must implement all required methods.

# CustomStringConvertible

# Printing with CustomStringConvertible

```
let string = "Hello, world!"  
print(string) // Hello, world!
```

```
let number = 42  
print(number) // 42
```

```
let boolean = false  
print(boolean) // false
```

# Printing with CustomStringConvertible

```
class Shoe {
  let color: String
  let size: Int
  let hasLaces: Bool

  init(color: String, size: Int, hasLaces: Bool) {
    ...
  }
}

let myShoe = Shoe(color: "Black", size: 12, hasLaces: true)
print(myShoe) // __lldb_expr_1.Shoe
```

```
class Shoe: CustomStringConvertible {  
    let color: String  
    let size: Int  
    let hasLaces: Bool  
  
    init(color: String, size: Int, hasLaces: Bool) {  
        ...  
    }  
  
}
```

```
class Shoe: CustomStringConvertible {  
    let color: String  
    let size: Int  
    let hasLaces: Bool  
  
    init(color: String, size: Int, hasLaces: Bool) {  
        ...  
    }  
  
    var description: String {  
        return "Shoe(color: \(color), size: \(size), hasLaces:  
\((hasLaces))"  
    }  
}
```

```
let myShoe = Shoe(color: "Black", size: 12, hasLaces: true)
print(myShoe) // Shoe(color: Black, size: 12, hasLaces: true)
```

# Equatable



# Comparing information with Equatable

```
struct Employee {  
    let firstName: String  
    let lastName: String  
    let jobTitle: String  
    let phoneNumber: String  
}  
  
struct Company {  
    let name: String  
    let employees: [Employee]  
}
```

# Comparing information with Equatable

```
let currentEmployee = Session.currentEmployee
let selectedEmployee = Employee(firstName: "Adrien",
    lastName: "Humilière", jobTitle: "Mobile engineer",
    phoneNumber: "415-555-9293")

if currentEmployee == selectedEmployee {
    // Enable "Edit" button
}
```

# Comparing information with Equatable

```
struct Employee: Equatable {  
    let firstName: String  
    let lastName: String  
    let jobTitle: String  
    let phoneNumber: String  
  
    static func ==(lhs: Employee, rhs: Employee) -> Bool {  
        // Equality logic  
    }  
}
```

# Comparing information with Equatable

```
struct Employee: Equatable {  
    let firstName: String  
    let lastName: String  
    let jobTitle: String  
    let phoneNumber: String  
  
    static func ==(lhs: Employee, rhs: Employee) -> Bool {  
        return lhs.firstName == rhs.firstName && lhs.lastName ==  
rhs.lastName  
    }  
}
```

# Comparing information with Equatable

```
let currentEmployee = Employee(firstName: "Adrien",  
    lastName: "Humilière", jobTitle: "Mobile engineer",  
    phoneNumber: "415-555-9293")  
let selectedEmployee = Employee(firstName: "Adrien",  
    lastName: "Humilière", jobTitle: "Customer support",  
    phoneNumber: "417-436-7384")  
  
if currentEmployee == selectedEmployee {  
    // Enable "Edit" button  
}
```

# Comparing information with Equatable

```
struct Employee: Equatable {  
    let firstName: String  
    let lastName: String  
    let jobTitle: String  
    let phoneNumber: String  
  
    static func ==(lhs: Employee, rhs: Employee) -> Bool {  
        return lhs.firstName == rhs.firstName && lhs.lastName ==  
rhs.lastName && lhs.jobTitle == rhs.jobTitle &&  
lhs.phoneNumber == rhs.phoneNumber  
    }  
}
```

# Comparable

# Sorting information with Comparable

```
let employee1 = Employee(firstName: "Ben", lastName: "Atkins")
let employee2 = Employee(firstName: "Vera", lastName: "Carr")
let employee3 = Employee(firstName: "Grant", lastName: "Phelps")
let employee4 = Employee(firstName: "Sang", lastName: "Han")

let employees = [employee1, employee2, employee3, employee4]
```



```
struct Employee: Equatable, Comparable {  
    let firstName: String  
    let lastName: String  
    let jobTitle: String  
    let phoneNumber: String  
  
    static func ==(lhs: Employee, rhs: Employee) -> Bool {  
        return ...  
    }  
  
    static func < (lhs: Employee, rhs: Employee) -> Bool {  
        return lhs.lastName < rhs.lastName  
    }  
}
```

```
let employees = [employee1, employee2, employee3, employee4,
employee5]

let sortedEmployees = employees.sorted(by:<)

for employee in sortedEmployees {
    print(employee)
}

// Employee(firstName: "Ben", lastName: "Atkins")
// Employee(firstName: "Vera", lastName: "Carr")
// Employee(firstName: "Sang", lastName: "Han")
// Employee(firstName: "Grant", lastName: "Phelps")
```

```
let employees = [employee1, employee2, employee3, employee4,
employee5]

let sortedEmployees = employees.sorted(by:>)

for employee in sortedEmployees {
    print(employee)
}

// Employee(firstName: "Grant", lastName: "Phelps")
// Employee(firstName: "Sang", lastName: "Han")
// Employee(firstName: "Vera", lastName: "Carr")
// Employee(firstName: "Ben", lastName: « Atkins")
```

# Codable

# Encoding and decoding objects with Codable

```
struct Employee: Equatable, Comparable, Codable {  
    var firstName: String  
    var lastName: String  
    var jobTitle: String  
    var phoneNumber: String  
  
    ...  
}
```

# Encoding and decoding objects with Codable

```
let ben = Employee(firstName: "Ben", lastName: "Atkins",
                    jobTitle: "Front Desk",
                    phoneNumber: "415-555-7767")

let jsonEncoder = JSONEncoder()
if let jsonData = try? jsonEncoder.encode(ben),
    let jsonString = String(data: jsonData, encoding: .utf8) {
    print(jsonString)
}
```

```
{"firstName": "Ben", "lastName": "Atkins", "jobTitle": "Front
Desk", "phoneNumber": "415-555-7767"}
```

# Protocol creation

# Creating a protocol

```
protocol FullyNamed {  
    var fullName: String { get }  
  
    func sayFullName()  
}  
  
struct Person: FullyNamed {  
    var firstName: String  
    var lastName: String  
}
```



# Creating a protocol

```
struct Person: FullyNamed {  
    var firstName: String  
    var lastName: String  
  
    var fullName: String {  
        return "\(firstName) \(lastName)"  
    }  
  
    func sayFullName() {  
        print(fullName)  
    }  
}
```

# Delegation

# Delegation

Enables a class or structure to hand off responsibilities to an instance of another type

```
protocol ButtonDelegate {  
    func userTappedButton(_ button: Button)  
}  
  
class GameController: ButtonDelegate {  
    func userTappedButton(_ button: Button) {  
        print("User tapped the \(button.title) button.")  
    }  
}
```

# Delegation

```
class Button {  
    let title: String  
    var delegate: ButtonDelegate? // Add a delegate property  
  
    init(title: String) {  
        self.title = title  
    }  
  
    func tapped() {  
        self.delegate?.userTappedButton(self)  
        // If the delegate exists, call the delegate  
        // function `userTappedButton` on the delegate  
    }  
}
```

# Delegation

```
let startButton = Button(title: "Start Game")  
let gameController = GameController()  
startButton.delegate = gameController  
  
startButton.tapped()
```



[adhumi.fr/teaching](http://adhumi.fr/teaching)

**Credentials**  
m2sar / sarM2