Unit 4—Lesson 1: 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.

Printing with CustomStringConvertible

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

let number = 42
print(number)

let boolean = false
print(boolean)
```

```
Hello, world!
42
false
```

Printing with CustomStringConvertible

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

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

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) {
        self.color = color
        self.size = size
        self.hasLaces = hasLaces
    }




}
```

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

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

    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)
```

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: "Jacob", lastName: "Edwards",
                                jobTitle: "Marketing Director", 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 {
    // Logic that determines if the value on the left hand side and right hand side are equal
  }

}
```

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: "Jacob", lastName: "Edwards",
    jobTitle: "Industrial Designer", phoneNumber: "415-555-7766")
let selectedEmployee = Employee(firstName: "Jacob", lastName: "Edwards",
    jobTitle: "Marketing Director", 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 {
      return lhs.firstName == rhs.firstName && lhs.lastName == rhs.lastName
             && lhs.jobTitle == rhs.jobTitle && lhs.phoneNumber == rhs.phoneNumber
  }
}
```

Sorting information with Comparable

```
let employee1 = Employee(firstName: "Ben", lastName: "Atkins", jobTitle: "Front Desk",
phoneNumber: "415-555-7767")
let employee2 = Employee(firstName: "Vera", lastName: "Carr", jobTitle: "CEO", phoneNumber:
"415-555-7768")
let employee3 = Employee(firstName: "Grant", lastName: "Phelps", jobTitle: "Senior Manager",
phoneNumber: "415-555-7770")
let employee4 = Employee(firstName: "Sang", lastName: "Han", jobTitle: "Accountant",
phoneNumber: "415-555-7771")
let employee5 = Employee(firstName: "Daren", lastName: "Estrada", jobTitle: "Sales Lead",
phoneNumber: "415-555-7772")

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

```
struct Employee: Equatable, Comparable {
  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
  }

  static func < (lhs: Employee, rhs: Employee) -> Bool {
    return lhs.lastName < rhs.lastName
  }
}
```

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

let sortedEmployees = employees.sorted(by:<)

for employee in sortedEmployees {
  print(employee)
}
```

```
Employee(firstName: "Ben", lastName: "Atkins", jobTitle: "Front Desk", phoneNumber: "415-555-7767")
Employee(firstName: "Vera", lastName: "Carr", jobTitle: "CEO", phoneNumber: "415-555-7768")
Employee(firstName: "Daren", lastName: "Estrada", jobTitle: "Sales Lead", phoneNumber: "415-555-7772")
Employee(firstName: "Sang", lastName: "Han", jobTitle: "Accountant", phoneNumber: "415-555-7771")
Employee(firstName: "Grant", lastName: "Phelps", jobTitle: "Senior Manager", phoneNumber: "415-555-7770")
```

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

let sortedEmployees = employees.sorted(by:>)

for employee in sortedEmployees {
  print(employee)
}
```

```
Employee(firstName: "Grant", lastName: "Phelps", jobTitle: "Senior Manager", phoneNumber: "415-555-7770")
Employee(firstName: "Sang", lastName: "Han", jobTitle: "Accountant", phoneNumber: "415-555-7771")
Employee(firstName: "Daren", lastName: "Estrada", jobTitle: "Sales Lead", phoneNumber: "415-555-7772")
Employee(firstName: "Vera", lastName: "Carr", jobTitle: "CEO", phoneNumber: "415-555-7768")
Employee(firstName: "Ben", lastName: "Atkins", jobTitle: "Front Desk", phoneNumber: "415-555-7767")
```

Encoding and decoding objects with Codable

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

    static func < (lhs: Employee, rhs: Employee) -> Bool {
        return lhs.lastName < rhs.lastName
    }
}
```

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"}
```

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

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

GameController example (continued)

```
class Button {
  let title: String
  var delegate: ButtonDelegate? // Add a delegate property to the Button

  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

GameController example (continued)

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

startButton.tapped()
```

```swift
class Button {
  let title: String
  var delegate: ButtonDelegate? // Add a delegate property to the Button

  init(title: String) {
    self.title = title
  }

  func tapped() {
    self.delegate?.userTappedButton(self) // If the delegate exists, call the delegate
                                          // function `userTappedButton` on the delegate
  }
}

class GameController: ButtonDelegate {

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

Delegation
MusicController example

```
let musicController = MusicController()

let startMusicButton = Button(title: "Play")
startMusicButton.delegate = musicController
let stopMusicButton = Button(title: "Pause")
stopMusicButton.delegate = musicController
```

```
class MusicController: ButtonDelegate {

  func playSong(_ song: Song) {
   print("Now playing \(song.title)")
  }

  func pauseSong() {
   print("Paused current song.")
  }

  func userTappedButton(_ button: Button) {
    if button.title == "Play" {
      playSong(Playlist.songs.first)
    } else if button.title == "Stop" {
      pauseSong()
    }
  }
}
```

Unit 4—Lesson 1
Lab: Protocols

Open and complete the exercises in `Lab - Protocols.playground`