import UIKit  
//: # Structures and Classes  
//: ### Swift structures and classes are much closer in functionality than in other languages  
//: ## Comparing Structures and Classes  
/\*:  
 ### Common things in structures and classes in Swift :  
 \* Define properties to store values  
 \* Define methods to provide functionality  
 \* Define subscripts to provide access to their values using subscript syntax  
 \* Define initializers to set up their initial state  
 \* Be extended to expand their functionality beyond a default implementation  
 \* Conform to protocols to provide standard functionality of a certain kind  
\*/  
/\*:  
 ### Additional capabilities of classes:  
 \* Inheritance enables one class to inherit the characteristics of another.  
 \* Type casting enables you to check and interpret the type of a class instance at runtime.  
 \* Deinitializers enable an instance of a class to free up any resources it has assigned.  
 \* Reference counting allows more than one reference to a class instance.  
\*/  
//: ### Guideline: As a general guideline, prefer structures and enumerations  
//: ### Definition Syntax  
/\*:  
 ### Syntax:  
 \*struct SomeStructure {\*  
   
 \*\/\/ structure definition goes here\*  
   
 \*}\*  
   
 \*class SomeClass {\*  
   
 \*\/\/ class definition goes here\*  
   
 \*}\*  
\*/  
//: ### an example of a structure definition and a class definition:  
struct Resolution {  
 var width = 0  
 var height = 0  
}  
class VideoMode {  
 var resolution = Resolution()  
 var interlaced = false  
 var frameRate = 0.0  
 var name: String?  
}  
//: ### The syntax for creating instances:  
let someResolution = Resolution()  
let someVideoMode = VideoMode()  
//: ### Accessing Properties  
//: ### Using \*dot syntax\*  
print("The width of someResolution is \(someResolution.width)")  
// Prints "The width of someResolution is 0"  
//: ### Accessing subproperties  
print("The width of someVideoMode is \(someVideoMode.resolution.width)")  
// Prints "The width of someVideoMode is 0"  
//: ### assign a new value to a variable property:  
someVideoMode.resolution.width = 1280  
print("The width of someVideoMode is now \(someVideoMode.resolution.width)")  
// Prints "The width of someVideoMode is now 1280"  
//: ## Memberwise Initializers for Structure Types  
let vga = Resolution(width: 640, height: 480)  
//: ### Note: Unlike structures, class instances don’t receive a default memberwise initializer  
//: ## NOTE: Structures and Enumerations Are Value Types  
let hd = Resolution(width: 1920, height: 1080)  
var cinema = hd  
cinema.width = 2048  
print("cinema is now \(cinema.width) pixels wide")  
// Prints "cinema is now 2048 pixels wide"  
  
print("hd is still \(hd.width) pixels wide")  
// Prints "hd is still 1920 pixels wide"  
/\*:  
 ![StructClass](struct.png)  
 \*/  
//: ### The same behavior applies to enumerations:  
enum CompassPoint {  
 case north, south, east, west  
 mutating func turnNorth() {  
 self = .north  
 }  
}  
var currentDirection = CompassPoint.west  
let rememberedDirection = currentDirection  
currentDirection.turnNorth()  
  
print("The current direction is \(currentDirection)")  
print("The remembered direction is \(rememberedDirection)")  
// Prints "The current direction is north"  
// Prints "The remembered direction is west"  
//: ## Classes Are Reference Types  
//: ### Rather than a copy, a reference to the same existing instance is used  
let tenEighty = VideoMode()  
tenEighty.resolution = hd  
tenEighty.interlaced = true  
tenEighty.name = "1080i"  
tenEighty.frameRate = 25.0  
  
let alsoTenEighty = tenEighty  
alsoTenEighty.frameRate = 30.0  
/\*:  
 ![StructClass](class.png)  
 \*/  
print("The frameRate property of tenEighty is now \(tenEighty.frameRate)")  
// Prints "The frameRate property of tenEighty is now 30.0"  
//: ### Note that tenEighty and alsoTenEighty are declared as constants, rather than variables  
//: ### \*tenEighty\* and \*alsoTenEighty\* themselves don’t “store” the VideoMode instance—instead, they both refer to a VideoMode instance  
//: ## Identity Operators  
//: ### Use these operators to check whether two constants or variables refer to the same single instance:  
/\*:  
 ### Reminder:  
 \* Identical to (===) : two constants or variables of class type refer to exactly the same class instance  
 \* Not identical to (!==)  
 \* Equal to (==) : two instances are considered equal or equivalent in value  
 \* Not equal to (!=)  
\*/  
if tenEighty === alsoTenEighty {  
 print("tenEighty and alsoTenEighty refer to the same VideoMode instance.")  
}  
// Prints "tenEighty and alsoTenEighty refer to the same VideoMode instance."