import UIKit  
//: # Extensions  
/\*:  
 ### Extensions in Swift can:  
 \* • Add computed instance properties and computed type properties  
 \* • Define instance methods and type methods  
 \* • Provide new initializers  
 \* • Define subscripts  
 \* • Define and use new nested types  
 \* • Make an existing type conform to a protocol  
\*/  
//: ## Extension Syntax  
//: ### Declare extensions with the extension keyword:  
//: extension SomeType {  
// new functionality to add to SomeType goes here  
//: }  
//: ### An extension can extend an existing type to make it adopt one or more protocols.  
//:extension SomeType: SomeProtocol, AnotherProtocol {  
 // implementation of protocol requirements goes here  
//:}  
//: ## Computed Properties  
extension Double {  
 var km: Double { return self \* 1\_000.0 }  
 var m: Double { return self }  
 var cm: Double { return self / 100.0 }  
 var mm: Double { return self / 1\_000.0 }  
 var ft: Double { return self / 3.28084 }  
}  
let oneInch = 25.4.mm  
print("One inch is \(oneInch) meters")  
// Prints "One inch is 0.0254 meters"  
let threeFeet = 3.ft  
print("Three feet is \(threeFeet) meters")  
// Prints "Three feet is 0.914399970739201 meters"  
//: ### the names of these properties can be appended to a floating-point literal value with dot syntax  
//: ### These properties are read-only computed properties, and so they are expressed without the get keyword, for brevity  
//: ### we can use it in a mathematical calculation:  
let aMarathon = 42.km + 195.m  
print("A marathon is \(aMarathon) meters long")  
// Prints "A marathon is 42195.0 meters long"  
//: ## Initializers  
//: ### Extensions can add new initializers to existing types  
//: ### Extensions can add new convenience initializers to a class, but they cannot add new designated initializers or deinitializers to a class  
//: ### If you use an extension to add an initializer to a value type that provides default values for all of its stored properties and does not define any custom initializers, you can call the default initializer and memberwise initializer for that value type from within your extension’s initializer  
struct Size {  
 var width = 0.0, height = 0.0  
}  
struct Point {  
 var x = 0.0, y = 0.0  
}  
struct Rect {  
 var origin = Point()  
 var size = Size()  
}  
let defaultRect = Rect()  
let memberwiseRect = Rect(origin: Point(x: 2.0, y: 2.0),  
 size: Size(width: 5.0, height: 5.0))  
  
extension Rect {  
 init(center: Point, size: Size) {  
 let originX = center.x - (size.width / 2)  
 let originY = center.y - (size.height / 2)  
 self.init(origin: Point(x: originX, y: originY), size: size)  
 }  
}  
let centerRect = Rect(center: Point(x: 4.0, y: 4.0),  
 size: Size(width: 3.0, height: 3.0))  
// centerRect's origin is (2.5, 2.5) and its size is (3.0, 3.0)