import UIKit  
  
//: # Methods  
//: ### Methods are functions that are associated with a particular type  
//: ### Classes, structures, and enumerations can all define instance methods  
//: ### Classes, structures, and enumerations can also define type methods  
//: ## Instance Methods  
//: ### Instance methods are functions that belong to instances of a particular class, structure, or enumeration.  
//: ### An instance method has implicit access to all other instance methods and properties of that type  
class Counter {  
 var count = 0  
 func increment() {  
 count += 1  
 }  
 func increment(by amount: Int) {  
 count += amount  
 }  
 func reset() {  
 count = 0  
 }  
}  
//: ### Calling methods:  
let counter = Counter()  
// the initial counter value is 0  
counter.increment()  
// the counter's value is now 1  
counter.increment(by: 5)  
// the counter's value is now 6  
counter.reset()  
// the counter's value is now 0  
//: ## The self Property  
//: ### Every instance of a type has an implicit property called self  
//: ### equivalent to the instance itself  
//: ### The increment() method in the example above could have been written like this:  
/\*:  
func increment() {  
 self.count += 1  
}  
\*/  
//: ### when the name of the method parameter and the name of the instance property are the same use the self property to distinguish between the parameter name and the property name.  
struct SimplePoint {  
 var x = 0.0, y = 0.0  
 func isToTheRightOf(x: Double) -> Bool {  
 return self.x > x  
 }  
}  
let somePoint = SimplePoint(x: 4.0, y: 5.0)  
if somePoint.isToTheRightOf(x: 1.0) {  
 print("This point is to the right of the line where x == 1.0")  
}  
// Prints "This point is to the right of the line where x == 1.0"  
//: ## Modifying Value Types from Within Instance Methods  
//: ### Structures and enumerations are value types. By default, the properties of a value type cannot be modified from within its instance methods.  
//: ### if you need to modify the properties of your structure or enumeration within a particular method, you can opt in to mutating behavior for that method  
//: \* keyword : mutating  
struct Point {  
 var x = 0.0, y = 0.0  
 mutating func moveBy(x deltaX: Double, y deltaY: Double) {  
 x += deltaX  
 y += deltaY  
 }  
}  
var anotherPoint = Point(x: 1.0, y: 1.0)  
anotherPoint.moveBy(x: 2.0, y: 3.0)  
print("The point is now at (\(anotherPoint.x), \(anotherPoint.y))")  
// Prints "The point is now at (3.0, 4.0)"  
//: ### you cannot call a mutating method on a constant of structure type  
let fixedPoint = Point(x: 3.0, y: 3.0)  
//: fixedPoint.moveBy(x: 2.0, y: 3.0)  
// this will report an error  
//: ## Assigning to self Within a Mutating Method  
//: ### Mutating methods can assign an entirely new instance to the implicit self property  
struct Point2 {  
 var x = 0.0, y = 0.0  
 mutating func moveBy(x deltaX: Double, y deltaY: Double) {  
 self = Point2(x: x + deltaX, y: y + deltaY)  
 }  
}  
//: ## Mutating methods for enumerations  
enum TriStateSwitch {  
 case off, low, high  
 mutating func next() {  
 switch self {  
 case .off:  
 self = .low  
 case .low:  
 self = .high  
 case .high:  
 self = .off  
 }  
 }  
}  
var ovenLight = TriStateSwitch.low  
ovenLight.next()  
// ovenLight is now equal to .high  
ovenLight.next()  
// ovenLight is now equal to .off  
//: ## Type Methods  
//: ### You can define methods that are called on the type itself  
//: \* static keyword before the method’s func keyword  
//: \* Classes may also use the class keyword to allow subclasses to override the superclass’s implementation of that method  
/\*:  
 ### calling type methods  
   
 class SomeClass {  
   
 class func someTypeMethod() {  
   
 // type method implementation goes here  
   
 }  
   
 }  
   
 SomeClass.someTypeMethod()  
\*/