巨匠線上真人

iOS行動程式基礎開發上架

swift：swift的方法

本堂教學重點

1. **實體方法**
   - self屬性
   - 修改值類型屬性的方法
   - 在mutating 方法內，使用self接收值
2. **類型方法**
   - 定義簡式版的Setter
   - 唯讀的計算屬性

1.實體方法

```
class Counter {
    var count = 0
    func increment() {
        count += 1
    }
    func increment(by amount: Int) {
        count += amount
    }
    func reset() {
        count = 0
    }
}
```

```
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
```

1.實體方法

self屬性

```
•   func increment() {
•       self.count += 1
•   }
•

•   truct Point {
•       var x = 0.0, y = 0.0
•       func isToTheRightOf(x: Double) -> Bool {
•           return self.x > x
•       }
•   }
•   let somePoint = Point(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"
•
```

1.實體方法

修改值類型屬性的方法

```
•   struct Point {
•       var x = 0.0, y = 0.0
•       mutating func moveBy(x deltaX: Double, y deltaY: Double) {
•           x += deltaX
•           y += deltaY
•       }
•   }
•   var somePoint = Point(x: 1.0, y: 1.0)
•   somePoint.moveBy(x: 2.0, y: 3.0)
•   print("The point is now at (\(somePoint.x), \(somePoint.y))")
•   // Prints "The point is now at (3.0, 4.0)"
•
```

```
•   let fixedPoint = Point(x: 3.0, y: 3.0)
•   fixedPoint.moveBy(x: 2.0, y: 3.0)
•   // this will report an error
```

1.實體方法

在mutating 方法內，使用self接收值

```
struct Point {
    var x = 0.0, y = 0.0
    mutating func moveBy(x deltaX: Double, y deltaY: Double) {
        self = Point(x: x + deltaX, y: y + deltaY)
    }
}
```

```
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
```

2.類型方法

```
class SomeClass {
    class func someTypeMethod() {
        // type method implementation goes here
    }
}
SomeClass.someTypeMethod()
```

2.類型方法

```swift
struct LevelTracker {
    static var highestUnlockedLevel = 1
    var currentLevel = 1

    static func unlock(_ level: Int) {
        if level > highestUnlockedLevel { highestUnlockedLevel = level }
    }

    static func isUnlocked(_ level: Int) -> Bool {
        return level <= highestUnlockedLevel
    }

    @discardableResult
    mutating func advance(to level: Int) -> Bool {
        if LevelTracker.isUnlocked(level) {
            currentLevel = level
            return true
        } else {
            return false
        }
    }
}
```

2.類型方法

```
class Player {
    var tracker = LevelTracker()
    let playerName: String
    func complete(level: Int) {
        LevelTracker.unlock(level + 1)
        tracker.advance(to: level + 1)
    }
    init(name: String) {
        playerName = name
    }
}
```

2.類型方法

```
var player = Player(name: "Argyrios")
player.complete(level: 1)
print("highest unlocked level is now \(LevelTracker.highestUnlockedLevel)")
// Prints "highest unlocked level is now 2"


player = Player(name: "Beto")
if player.tracker.advance(to: 6) {
    print("player is now on level 6")
} else {
    print("level 6 has not yet been unlocked")
}
// Prints "level 6 has not yet been unlocked"

```