10 CHAPTER

屬性與方法

屬性 (property) 是與特定類別、結構或列舉的資料變數。相當於其它程式語言如 C++或 Java 的資料成員 (data member)。但 Swift 的屬性又可加以分類為儲存型屬性與計算型屬性。本章除了要探討這兩個主題外，還討論了屬性觀察者 (property observer) 及型態屬性 (type property)。

在類別、結構與列舉除了定義屬性外，還可以定義方法 (method) 。方法又可以分成實例方法 (instance method) 與型態方法 (type method)。除了上述主題，也會論及方法的區域與外部參數名稱、self 屬性及如何在方法內修改參數值。其實方法是物件導向程式設計的說法，它相當於一般傳統程式語言，如 C 語言的函式，都是用於解決某一問題有限步驟，有一簡單的辨別的方式，可以將類別、結構與列舉內的函式稱之為方法。

10.1 儲存型屬性

前面章節所談到的不管在類別或結構內的變數或常數皆為儲存型屬性 (stored property) 。我們以一範例來加以解釋。如下所示：

範例程式

```
01  // store property
02  struct Circle {
03      var x, y: Int
04      var radius = 10
05  }
```

```
06
07  let cirObj = Circle(x: 10, y: 10, radius: 11)
08
09  print("Circle:")
10  print("x: \(cirObj.x), y: \(cirObj.y)")
11  print("radius: \(cirObj.radius)")
```

輸出結果

```
Circle:
x: 10, y: 10
radius: 11
```

程式中的 x、y，以及 radius 稱之為結構 Circle 的屬性變數。x 與 y 資料型態為 Int，且 radius 也是整數，初始值為 10。這三個變數的屬性皆為儲存型屬屬性。當下一敘述

```
let cirObj = Circle(x: 10, y: 10, radius: 11)
```

執行後，定義 cirObj 物件，其 x、y、radius 分別設定初值為 10、10、11。

10.2 計算型屬性

計算型屬性 (computed property) 不是真正儲存一值，取而代之的是提供用以擷取的 getter 和選擇性的 setter，用以間接設定某一屬性值。如以下範例所示：

範例程式

```
01  // computed property
02  struct Point {
03      var x = 0.0, y = 0.0
04  }
05
06  struct Side {
07      var length = 0.0
08  }
09  struct Square {
10      var originPoint = Point()
11      var side = Side()
12      var center: Point {
```

```
13      get {
14          let centerPointX =  originPoint.x + side.length / 2
15          let centerPointY =  originPoint.y + side.length / 2
16          return Point(x: centerPointX, y: centerPointY)
17      }
18
19      set(newCenter) {
20          originPoint.x = newCenter.x - side.length / 2
21          originPoint.y = newCenter.y - side.length / 2
22       }
23       }
24  }
25
26  var Obj = Square(originPoint: Point(x: 0.0, y: 0.0), side: Side(length: 10))
27
28  // call getter
29  print("center x: \(Obj.center.x), y: \(Obj.center.y)")
30
31  // call setter
32  Obj.center = Point(x: 12, y: 12)
33  print("original x: \(Obj.originPoint.x), y: \(Obj.originPoint.y)")
```

輸出結果

```
center x: 5.0, y: 5.0
original x: 7.0, y: 7.0
```

此範例程式利用 get 與 set 函式，來間接擷取屬性值與設定屬性值。我們利用 get 函式得到原來的中心點，再利用 set 函式設計新的中心點，之後印出原點。

其中 get 函式如下：

```
get {
    let centerPointX =  originPoint.x + side.length / 2
    let centerPointY =  originPoint.y + side.length / 2
    return Point(x: centerPointX, y: centerPointY)
}
```

表示將原點加上正方形長度的一半 (0 + 10/2, 0 + 10/2)即為中心點，並加以回傳，亦即(5, 5)。而 set 函式是接收一新的中心點 newCenter 為其參數，之後計算新的原點為何，如下所示：

```
set(newCenter) {
    originPoint.x = newCenter.x - side.length / 2
    originPoint.y = newCenter.y - side.length / 2
}
```

新的原點是由新的中心點減去正方形長度的一半而得。其示意圖如下：

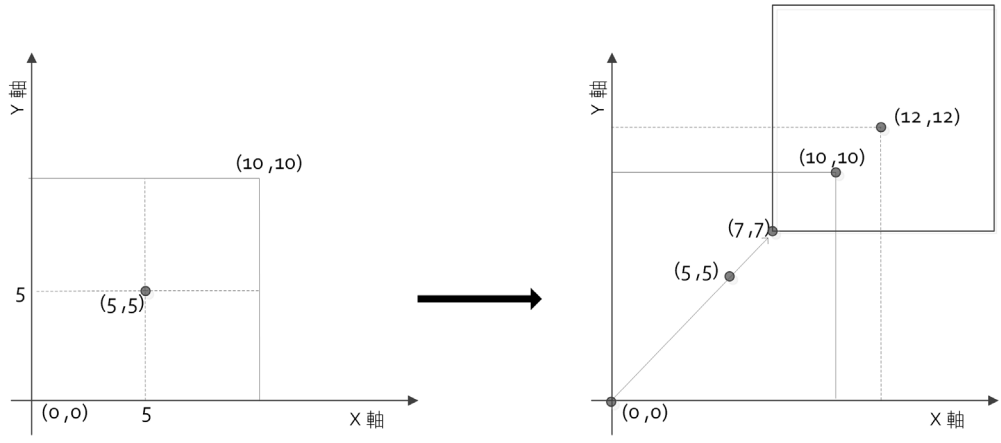

圖 10-5

最後利用下一敘述呼叫 get 函式，印出正方形的中心點。

```
// call getter
print("center x: \(Obj.center.x), y: \(Obj.center.y)")
```

及利用下一敘述呼叫 set 函式，設定新的中心點，然後計算新的原點，最後將其印出

```
// call setter
Obj.center = Point(x: 12, y: 12)
print("original x: \(Obj.originPoint.x), y: \(Obj.originPoint.y)")
```

此為提供用以擷取新的原點 (12-10/2, 12-10/2)，亦即 (7, 7)。

10.2.1 速記 setter 宣告

速記計算型屬性(shorthand setter declaration)將程式中 set 的參數 newCenter 改用預設值 newValue。如下所示：

範例程式

```
// computer property
struct Point {
    var x = 0.0, y = 0.0
}

struct Side {
    var length = 0.0
}
struct Square {
    var originPoint = Point()
    var side = Side()
    var center: Point {
    get {
        let centerPointX =  originPoint.x + side.length / 2
        let centerPointY =  originPoint.y + side.length / 2
        return Point(x: centerPointX, y: centerPointY)
     }

    set {
        originPoint.x = newValue.x - side.length / 2
        originPoint.y = newValue.y - side.length / 2
     }
     }
}

var Obj = Square(originPoint: Point(x: 0.0, y: 0.0), side: Side(length: 10))

// call getter
print("center x: \(Obj.center.x), y: \(Obj.center.y)")

// call setter
Obj.center = Point(x: 12, y: 12)
print("original x: \(Obj.originPoint.x), y: \(Obj.originPoint.y)")
```

此程式與上一節程式唯一的差別在於，我們使用 setter 的預設值 newValue 取代上一程式的 newCenter，並且將 set 函式的參數省略，直接在 set 函式主體使用 newValue 即可。此程式的輸出結果同上。

10.2.2 唯讀計算型屬性

唯讀計算型屬性(read-only computed property)表示只有 get 函式，但沒有 set 函式，如下範例程式所示：

範例程式

```
// read-only computed property
struct cuboid {
    var width = 0.0, height = 0.0, depth = 0.0
    var volume: Double {
    get{
        return width * height * depth
    }
    }
}
let oneCuboid = cuboid(width: 2.0, height: 3.0, depth: 4.0)
print("Volume is \(oneCuboid.volume)")
```

輸出結果

```
Volume is 24.0
```

程式中只有 get 函式，用以計算圓柱體體積，並加以回傳。也可以將程式中 get 的關鍵字省略。如下範例程式所示：

範例程式

```
// read-only computed property
struct cuboid {
    var width = 0.0, height = 0.0, depth = 0.0
    var volume: Double {
        return width * height * depth
    }
}
let oneCuboid = cuboid(width: 2.0, height: 3.0, depth: 4.0)
print("Volume is \(oneCuboid.volume)")
```

輸出結果同上。

10.3 屬性的觀察者

屬性的觀察者 (property observer) 將觀察和反應屬性值的變化。每一次屬性值被設定時，將會呼叫屬性觀察者，即使新的設定值和目前的值是一樣。

屬性觀察者有二個函式，一為 willSet，它在屬性值儲存之前將被呼叫，而另一個 didSet 函式是新的值儲存後加以執行。我們以一範例說明之，如下所示：

範例程式

```
// properties observers
class YoursScore {
    var score: Int = 0 {
        willSet(newScore) {
            print("您的分數是 \(newScore)")
        }

        didSet {
            if score > oldValue {
                print("您進步了 \(score - oldValue) 分")
            } else {
                print("您退步了 \(oldValue - score) 分")
            }
        }
    }
}

let yourScore = YoursScore()
yourScore.score = 60
yourScore.score = 80
yourScore.score = 70
```

輸出結果

```
您的分數是 60
您進步了 60 分
您的分數是 80
```

```
您進步了 20 分
您的分數是 70
您退步了 10 分
```

willSet 接收一參數 newScore（第 4 行），然後將其印出，而第 8 行 didSet 是判斷 score 是否大於預設值 oldValue，分別印出進步幾分或是退步幾分。當第一個設定

```
yourScore.score = 60
```

執行前先處理 willSet，此時 newScore 是 60，再呼叫 didSet，由於 oldValue 是 0，所以 if 的判斷式為真，因此印出您進步了 60 分。接著

```
yourScore.score = 80
```

表示 score 為 80，而 oldValue 是上一次留下來的 60，所以執行 didSet 時，將印出您進步 20 分。最後

```
yourScore.score = 70
```

執行 didSet 時，將印出您退步了 10 分，因為此時的 oldValue 為 80，而 score 為 70。

其實程式中的 newScore 也可以預設值 newValue 取代之（下一範例程式第 5 行），此時 willSet 也就不必接收參數了，如下一範例所示：

範例程式

```
01  class YoursScore {
02      var score: Int = 0 {
03          // newScore 以 newValue 取代之
04          willSet {
05              print("您的分數是 \(newValue)")
06          }
07
08          didSet {
09              if score > oldValue {
10                  print("您進步了 \(score - oldValue) 分")
11              } else {
12                  print("您退步了 \(oldValue - score) 分")
13              }
14          }
```

```
    }
}

let yourScore = YoursScore()
yourScore.score = 60
yourScore.score = 80
yourScore.score = 70
```

輸出結果和上一範例相同。

10.4 型態屬性

類別、結構或列舉的屬性，一般是預設為實例屬性 (instance property) ，它必須以物件加以存取，因為是某一物件所擁有。而另一是型態屬性(type property)，它不必建立物件，而只要以類別、結構或是列舉名稱來呼叫即可。而如何形成型態屬性呢? 在結構或列舉是以 static 關鍵字加入於屬性的前面，這類似 C++ 的 static，其意義是共享或共用的意思，不是屬於任何一物件所有。如以下程式所示：

範例程式

```
// type property of struct
struct Rectangle {
    static var width = 20
    static var height = 30
    static var property: String {
        return "Rectangle: "
    }
}

print(Rectangle.property)
print("Width: \(Rectangle.width)")
print("Height: \(Rectangle.height)")
```

輸出結果

```
Rectangle:
Width: 20
Height: 30
```

上述的結構內的屬性前皆加上 static 關鍵字，所以它們是屬於型態屬性，因此不需要建立物件，只要以結構名稱呼叫即可。如上述的 Rectangle.property、Rectangle.width 以及 Rectangle.height。若是將 static 關鍵字刪除，則變為實例屬性（instance property），如下程式所示：

範例程式

```
struct Rectangle {
    var width = 20
    var height = 30
    var property: String {
        return "Rectangle: "
    }
}

let obj = Rectangle()
print(obj.property)
print("Width: \(obj.width)")
print("Height: \(obj.height)")
```

輸出結果同上。此時則必須先建立一 Rectangle 物件 obj，再利用 obj 存取其屬性。否則會有錯誤的訊息產生。

而類別的型態屬性則需在屬性前加上 class 關鍵字，但類別的型態屬性不可以用在儲存型屬性，如以下範例程式所示：

範例程式

```
// type property of class
class Rectangle {
    var width = 20
    var height = 30
    class var property: String {
        return "Rectangle: "
```

```
    }
}

let rectObj = Rectangle()
print(Rectangle.property)
print("Width: \(rectObj.width)")
print("Height: \(rectObj.height)")
```

輸出結果同上。第 5 行的 property 為型態屬性(因為多加了 class)。注意，width 和 height 是儲存型屬性，所以不能使用型態使用，而 property 則可以。

10.5 實例方法

實例方法 (instance method) 就是定義於類別或結構的方法，稱之為實例方法。實例方法必須使用物件來呼叫。我們以一範例程式來說明：

範例程式

```
// instance method
class Circle {
    var radius = 0.0
    //instance method
    func getArea() -> Double {
        return radius * radius * 3.14159
    }
    func setRadius(r: Double) {
        radius = r
    }
}

let circleObject = Circle()
circleObject.setRadius(r: 10)
let circleArea = circleObject.getArea()
print("圓面積: \(circleArea)")
```

輸出結果

```
圓面積: 314.159
```

程式中有兩個方法，分別是 getArea()

```
func getArea() -> Double {
    return radius * radius * 3.14159
}
```

此方法無參數但有回傳值，其型態為 Double。另一個方法為 setRadius(r:)

```
func setRadius(r: Double) {
    radius = r
}
```

此方法接收一參數，其型態為 Double，用以設定新半徑值。此方法沒有回傳值。

由於這兩個方法是實例方法，所以必須先建立一物件，如下所示：

```
let circleObject = Circle()
```

之後便可利用 circleObject 呼叫 getArea 方法和 setRadius 方法，如下所示：

```
circleObject.setRadius(r: 10)
let circleArea = circleObject.getArea()
```

分別設定圓的半徑為 10, 以及計算圓面積，並將它指定給 circleArea 常數名稱。

10.5.1 方法的參數名稱

在函式那一章曾提過外部參數名稱，其實就是讓程式可讀性提高。如以下的範例是設定一 Rectangle 類別，其中有兩個屬性及兩個方法，分別是 getArea 用以得到矩形面積，另一個是 setWidthAndHeight(w:h:)用以設定新的 width 與 height。

在 Swift 3 版本，方法參數名稱皆要在呼叫方法時加以列出。如以下程式所示：

範例程式

```
01  class Rectangle {
02      var width = 0.0
03      var height = 0.0
04      func getArea() -> Double {
05          return width * height
```

```
    }
    func setWidthAndHeight(w: Double, h: Double) {
        width = w
        height = h
    }
}
let rectObject = Rectangle()
rectObject.setWidthAndHeight(w: 10, h: 20)
let rectArea = rectObject.getArea()
print("矩形面積: \(rectArea)")
```

輸出結果

```
矩形面積: 200.0
```

在下一敘述中

```
rectObject.setWidthAndHeight(w: 10, h: 20)
```

參數名稱 w 和 h 不可以省略（第 13 行），否則會產生錯誤訊息。在 Swift 3 之前的版本，方法中的第一個參數名稱可以省略。當然您也可以強制加上外部參數名稱，如下範例程式所示：

範例程式

```
class Rectangle {
    var width = 0.0
    var height = 0.0
    func getArea() -> Double {
        return width * height
    }
    func setWidthAndHeight(width w: Double, height h: Double) {
        width = w
        height = h
    }
}
let rectObject = Rectangle()
rectObject.setWidthAndHeight(width: 10, height: 20)
let rectArea = rectObject.getArea()
print("矩形面積: \(rectArea)")
```

此時呼叫 setWidthAndHeight 方法時（第 13 行），必須將外部參數名稱 width 與 height 寫出，如下所示：

```
rectObject.setWidthAndHeight(width: 10, height: 20)
```

不過這方式目前較少人採用。因為每一個參數名稱在呼叫方法時皆要寫出，為什麼還要多此一舉再加上另一參數名稱，您說是嗎？

10.5.2 self 屬性

Swift 的 self 和 C++ 與 Java 的 this 關鍵字具有相同的意義，表示類別本身的意思。self 是自動產生的，不用您去宣告與定義。當方法所接收的參數名稱和要指定給類別、結構或列舉的屬性名稱相同時，就得使用 self 屬性，表示此為本身的屬性。如下範例程式所示：

範例程式

```
01  // using self
02  class Rectangle {
03      var width = 0.0
04      var height = 0.0
05      func getArea() -> Double {
06          return width * height
07      }
08      func setWidthAndHeight(w: Double, h: Double) {
09          self.width = w
10          self.height = h
11      }
12  }
13
14  let rectangleObject = Rectangle()
15  rectangleObject.setWidthAndHeight(w: 10, h: 20)
16  let totalArea = rectangleObject.getArea()
17  print("面積: \(totalArea)")
```

上面的範例中第 8 行的 setWidthAndHeight 方法，其實有無 self 都是可以的，如下所示：

```
func setWidthAndHeight(w: Double, h: Double) {
    width = w
    height = h
}
```

但若是參數名稱和指定的屬性名稱相同時，則必須藉用 self，否則無法運作。如下範例程式所示：

範例程式

```
01  // using self
02  class Rectangle {
03      var width = 0.0
04      var height = 0.0
05      func getArea() -> Double {
06          return width * height
07      }
08      func setWidthAndHeight(width: Double, height: Double) {
09          self.width = width
10          self.height = height
11      }
12  }
13
14  let rectangleObject = Rectangle()
15  rectangleObject.setWidthAndHeight(width: 10, height: 20)
16  let totalArea = rectangleObject.getArea()
17  print("面積: \(totalArea)")
```

setWidthAndHeight 方法中的第 9~10 行敘述必須將 self 加以寫出

```
func setWidthAndHeight(width: Double, height: Double) {
    self.width = width
    self.height = height
}
```

因為省略 self 將會造成無法辨認 width 與 height 是屬於參數的或是本身的屬性。

10.5.3 從實例方法內修改值型態

因為結構和列舉的屬性是屬於值型態 (value type)。預設上，值型態的屬性不可以在實例方法中被修改，若要執行修改的動作，必須在方法前加上 mutating 的關鍵字。如將本章最前面的那一個範例，將 class 改為 struct，此時 setRadius 方法前必須加上 mutating（第 8 行），否則會產生錯誤的訊息，如下所示：

範例程式

```
01  // mutating keyword
02  // 在 setRadius 函式前加上 mutating
03  struct Circle {
04      var radius = 0.0
05      func getArea() -> Double {
06          return radius * radius * 3.14159
07      }
08      mutating func setRadius(r: Double) {
09          radius = r
10      }
11  }
12
13  //物件一定要是 var
14  var circleObject = Circle()
15  circleObject.setRadius(r: 10)
16  let totalArea = circleObject.getArea()
17  print("面積: \(totalArea)")
```

輸出結果

```
面積: 314.159
```

由於類別是屬於參考型態，所以可以在方法內修改屬性值，也因此 mutating 只用於結構或是列舉。還有要注意的是 circleObject 一定要設為變數名稱，因為有更改類別的 radius 屬性值。

10.6 型態方法

在本章 10.4 節已探討過型態屬性，本節將討論型態方法(type method) 。其與實例方法的差異是，型態方法是共用的，有如 C++ 或 Java 的 static 方法。型態方法可以使用類別名稱或結構名稱來呼叫，不必定義類別或類別的物件來呼叫。

類別的型態方法是在 func 前加上 class 的關鍵字（第 5 行），而結構與列舉則是在 func 前加上 static 關鍵字。讓我們來看幾個範例。

範例程式

```
01  // type method of class using class
02  class Circle {
03      var radius = 0.0
04      // type method
05      class func printStar() {
06          print("********")
07      }
08      func getArea() -> Double {
09          return radius * radius * 3.14159
10      }
11      func setRadius(r: Double) {
12          radius = r
13      }
14  }
15
16  let circleObject = Circle()
17  Circle.printStar()
18  circleObject.setRadius(r: 10)
19  let totalArea = circleObject.getArea()
20  print("\(totalArea)")
```

輸出結果

```
*******
面積: 314.159
```

程式定義了類別 Circle，同時將 printStar 方法設定為型態方法，因為此方法前加上了 class 關鍵字。因此，在呼叫 printStar 方法時，可以使用類別名稱 Circle 直接呼叫（第 17 行）。而另兩個實例方法 getArea 與 setRadius 必須以 Circle 的物件 circleObject 呼叫（第 18-19 行）。

若將類別改以結構表示時，則定義型態方法是要加上 static 關鍵字，同時也必須在 setRadius(r:)方法前加上 mutating。順帶一提的是，定義 circleObject 物件必須是變數名稱。如下範例程式所示：

範例程式

```
01  // type method of structure using static
02  struct Circle {
03      var radius = 0.0
04      static func printStar() {
05          print("*********")
06      }
07      func getArea() -> Double {
08          return radius * radius * 3.14159
09      }
10      mutating func setRadius(r: Double) {
11          radius = r
12      }
13  }
14
15  var circleObject = Circle()
16  Circle.printStar()
17  circleObject.setRadius(r: 10)
18  let totalArea = circleObject.getArea()
19  print("面積: \(totalArea)")
```

型態方法在類別的屬性前加上 class，而在結構與列舉的屬性加上 static（第 4 行）。有一點要說明的是，型態屬性只能被型態方法所使用，否則會產生錯誤的訊息。如以下範例程式所示：

範例程式

```
01  // structure of type property and type method
02  // type property  只能被  type method 使用
03  struct Circle {
04      static var radius = 0.0
05      static func printstar() {
06          print("*******")
07      }
08
09      static func getArea() ->Double {
10          return radius * radius * 3.14159
11      }
12      static func setRadius(r: Double) {
13          radius = r
14      }
15  }
16
17  var circleObject = Circle()
18  Circle.printstar()
19  Circle.setRadius(r: 10)
20  let totalArea = Circle.getArea()
21  print("\(totalArea)")
```

從上一範例程式得知，static 的型態屬性（第 4 行）可以被型態方法（第 12 行）更改，所以不必加 mutating。也就是說 static 的型態方法不用再加上 mutating 的關鍵字。

自我練習題

1. 試問下列程式的輸出結果。

```
// computed property
struct Point {
    var x = 0.0, y = 0.0
}

struct Rectangle {
    var origin = Point()
    var width = 0.0, height = 0.0
    var center: Point {
    get{
        let centerX = origin.x + width / 2
        let centerY = origin.x + height / 2
        return Point(x: centerX, y:centerY)
    }

    set(newCenter) {
        origin.x = newCenter.x - width / 2
        origin.y = newCenter.y - height / 2
    }
    }
}

var square = Rectangle(origin: Point(x: 0.0, y: 0.0), width: 6, height: 6)
square.center = Point(x: 10.0, y: 10.0)
print("square.origin is now at \(square.origin.x), \(square.origin.y)")
```

2. 以下程式皆有些許的 bugs，請你來 debug 一下，順便測驗你對本章的了解程度。

(a)

```
// computer property
struct Point {
    var x = 0.0, y = 0.0
}

struct Side {
    var length = 0.0
}
```

```
    var originPoint = Point()
    var xandY = Side()
    var center: Point {
    get {
        let centerPointX =  originPoint.x + xandY.length / 2
        let centerPointY =  originPoint.y + xandY.length / 2
        return Point(x: centerPointX, y: centerPointY)
    }

    set {
        originPoint.x = newCenter.x - xandY.length / 2
        originPoint.y = newCenter.y - xandY.length / 2
    }
    }
}

var Obj = Square(originPoint: Point(x: 0.0, y: 0.0), side:Side(length: 10))

// call getter
println("center x: \(Obj.center.x), y: \(Obj.center.y)")

// call setter
Obj.center = Point(x: 12, y: 12)
print("original x: \(Obj.originPoint.x), y: \(Obj.originPoint.y)")
```

(b)

```
// properties observers
class YoursScore {

    var score: Int = 0 {

    willset(newScore) {
        print("您的分數是 \(newScore)")
    }

    didset {
        if score > oldValue {
            print("您進步了 \(score - oldValue) 分")
        } else {
            print("您退步了 \(oldValue - score) 分")
        }
    }
    }
}
```

```
let yourScore = YoursScore()
yourScore.score = 60
yourScore.score = 80
yourScore.score = 70
```

(c)

```
// type property
struct Rectangle {
    static var width = 20
    var height = 30
    var property: String {
        return"Rectangle: "
    }
}

print(Rectangle.property)
print("Width: \(Rectangle.width)")
print("Height: \(Rectangle.height)")
```

3. 下列程式是小蔡同學所撰寫，但有些錯誤訊息，聰明的你可否幫他除錯一下。

```
struct Circle {
    var radius = 0.0
    func getArea() -> Double {
        return radius * radius * 3.14159
    }
    func setRadius(r: Double) {
        radius = r
    }
}

var circleObject = Circle()
circleObject.setRadius(10)
let totalArea = circleObject.getArea()
print("面積: \(totalArea)")
```

4. 下列程式是小王同學所撰寫，但有些錯誤訊息，聰明的你可否幫他除錯一下。

```
struct Circle {
    var radius = 0.0
    func printStar() {
        print("*********")
    }
    func getArea() ->Double {
        return radius * radius * 3.14159
    }
    func setRadius(r: Double) {
        radius = r
    }
}

let circleObject = Circle()
Circle.printStar()
circleObject.setRadius(10)
let totalArea = circleObject.getArea()
print("面積: \(totalArea)")
```

5. 下列程式是小張同學所撰寫，但有些錯誤訊息，聰明的你可否幫他除錯一下。

```
struct Circle {
    static var radius = 0.0
    static func printStar() {
        print("*******")
    }

    func getArea() -> Double {
        return radius * radius * 3.14159
    }
    func setRadius(r: Double) {
        radius = r
    }
}

var circleObject = Circle()
Circle.printStar()
Circle.setRadius(r: 10)
```

```
Let totalArea = Circle.getArea()
print("\(totalArea)")
```

6. 下列程式是 Nancy 同學所撰寫，但有些錯誤訊息，聰明的你可否幫他除錯一下。

```
// structure of type property and type method
// type property  只能被  type method 使用

struct Circle {
    var radius = 0.0
    static func printStar() {
        print("*******")
    }

    static func getArea() ->Double {
        return radius * radius * 3.14159
    }
    static func setRadius(r: Double) {
        radius = r
    }
}

var circleObject = Circle()
Circle.printStar()
Circle.setRadius(10)
let totalArea = Circle.getArea()
```