(e)

```
func sumAndMean() -> (Int, Double) {
    let data = [1, 2, 3, 4, 5, 6, 7, 8 ,9, 10]
    var sum = 0, mean = 0
    for i in data {
        sum += i
    }
    mean = (sum) / (data.count)
    return (sum, mean)
}

let counter = sumAndMean()
print("sum=\(counter.sum), mean=\(counter.mean)")
```


8 CHAPTER 閉包

閉包(closure)是自我包含 (self-contained) 功能的區段，用以完成某項的任務。閉包可視為沒有名稱的函式。Swift 的閉包類似其它程式語言，如 C 語言的區段(block)，C++ 和 Objective-C 的拉姆達 (Lambda) 等等。

閉包可採用全域函式、巢狀函式，以及閉包運算式三種格式。以下將以閉包的運算式來加以說明。

8.1 閉包運算式

Swift 的閉包運算式 (closure expression) 是一清楚又簡潔的格式，其語法如下：

```
{ (parameters) -> return type in
    statements
}
```

閉包運算式語法可以使用常數參數、變數參數以及 inout 參數，但無法使用預設值。我們以範例來加以說明。

假設要將一陣列的資料由小至大排列，一般的撰寫方式如下：

範例程式

```
01 let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]
02 func ascending(a: Int, b: Int) -> Bool {
```

```
03      return  a < b
04  }
05  print("排序前的資料：")
06  for i in numbers {
07      print("\(i) ", terminator: "")
08  }
09  //  呼叫 sort 函式
10  var finished = numbers.sorted(by: ascending)
11
12  print("\n 排序後的資料：")
13  for i in finished {
14      print("\(i) ", terminator: "")
15  }
16  print("")
```

輸出結果

```
排序前的資料:
10 8 20 7 56 3 2 1 99
排序後的資料:
1 2 3 7 8 10 20 56 99
```

其中 ascending 是一函式（第 2-4 行），接收兩個 Int 的參數，而且回傳值為 Bool。當參數 a 小於參數 b 時回傳真。這表示第一個數字小於第二個數字，所以是由小至大排序。之後執行第 10 行敘述

```
var finished = numbers.sorted(by: ascending)
```

程式呼叫系統提供的 sorted 函式，此函式有一個參數，用以判斷是由小至大，或是由大至小的排序。最後排序的結果指定給 finished 陣列變數。

今將上述的寫法以閉包運算式方式撰寫，其程式如下所示：

範例程式

```
01  // closure
02  let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]
03
04  var finished = numbers.sorted(by: {(a: Int, b: Int) -> Bool in return a < b})
05  print("排序後的資料：")
```



```
06  for i in finished {
07      print("\(i) ", terminator: "")
08  }
09  print("")
```

輸出結果

```
排序後的資料:
1 2 3 7 8 10 20 56 99
```

其中的 sort 函式寫法如下：

```
numbers.sorted(by: {(a: Int, b: Int) -> Bool in return a < b})
```

此敘述將 ascending 函式以閉包運算式取代。

有關閉包運算式計有推論型態格式、明確的從單一運算式的閉包回傳、速記引數名稱，以及運算子函式等四種方式，我們將以範例一一解釋。

8.1.1 推論型態格式

其實 ascending 函式的型態為 (Int, Int) ->Bool，所以可使用推論型態(infer type)表示閉包運算式。因此，可將第 4 行 sorted 函式內的閉包運算式

```
{(a: Int, b: Int) -> Bool in return a < b}
```

簡化為

```
{(a, b) in return a < b}
```

完整的程式如下所示：

範例程式

```
01  // Inferred type from closure
02  let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]
03
04  var finished = numbers.sorted(by: {(a, b) in return a < b})
05  print("排序後的資料：")
06  for i in finished {
07      print("\(i) ", terminator: "")
08  }
09  print("")
```

輸出結果同上。

8.1.2 明確從單一運算式的閉包回傳

我們也可以從單一運算式的閉包回傳來表示閉包運算式。上一範例的閉包運算式可以下式表示。

```
{(a, b) in a < b }
```

完整程式如下所示：

範例程式

```
01  // Another type from closure
02  let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]
03
04  var finished = numbers.sorted(by: {(a, b) in a < b})
05  print("排序後的資料：")
06  for i in finished {
07      print("\(i) ", terminator: "")
08  }
09  print("")
```

輸出結果同上。和推論型態的差異是將 return 省略了（第 4 行）。

8.1.3 速記引數名稱

若要再簡單一點的話，可以速記引數名稱來表示，以上一範例程式來說，其閉包運算式可以下一敘述表示：

```
{$0 < $1})
```

看起來有沒有更簡單了。其完整的程式如下所示：

範例程式

```
01  // Shorthand argument names
02  let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]
03
04  var finished = numbers.sorted(by: {$0 < $1})
05  print("排序後的資料：")
06  for i in finished {
```



```
07      print("\(i) ", terminator: "")
08  }
09  print("")
```

輸出結果同上。$0 < $1 這樣的表示（第 4 行），您是否有感覺它是兩個參數的比較而已。

8.1.4 運算子函式

最後的閉包運算式是運算子函式，它是最簡的運算式，以上一範例來說，只要以運算子 < 來表示即可。程式如下所示：

範例程式

```
01  // operator function
02  let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]
03
04  var finished = numbers.sorted(by: <)
05  print("排序後的資料：")
06  for i in finished {
07      print("\(i) ", terminator: "")
08  }
09  print("")
```

輸出結果同上。程式中只以 < 表示（第 4 行），知其前者小於後者，若要由大至小，則以 > 表示之，表示前者大於後者。

8.2 尾隨閉包

上一節我們論及有關閉包運算式的方式，其實還有一種方式是當閉包運算式太長時，則可使用尾隨的閉包 (tailing closure) 。我們若將第一個範例程式的閉包運算式

```
var finished = numbers.sorted(by: {(a: Int, b: Int) -> Bool in return a < b})
```

改以尾隨的閉包表示的話，則程式如下所示：

```
var finished = numbers.sorted() {
    (a: Int, b: Int) -> Bool in return a < b }
```

可以看出閉包運算式脫離 sorted 函式的參數表示法，而是將閉包運算式緊接在其後面。

完整的程式，如下所示：

範例程式

```
01 // trailing closure
02 let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]
03
04 var finished = numbers.sorted() {(a: Int, b: Int) -> Bool in return a < b}
05 print("排序後資料: ")
06 for i in finished {
07     print("\(i) ", terminator: "")
08 }
09 print("")
```

輸出結果同上。也可以將它以速記引數名稱來表示，則情形如下：

範例程式

```
01 // trailing closure
02 let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]
03
04 var finished = numbers.sorted() {$0 < $1}
05 print("排序後資料: ")
06 for i in finished {
07     print("\(i) ", terminator: "")
08 }
09 print("")
```

輸出結果同上。

8.3 擷取數值

閉包可以存取周圍附近的常數和變數，因而閉包可以參考和修改在函式主體的變數與常數值。

巢狀函式 (nested function) 表示函式內部又有一函式，此也是閉包的一種表示方式。巢狀函式可以擷取(capture)任何位於外部函式的參數，而且可以擷取定義於外部函式的任何常數與變數。雖然全域函式也是閉包的一種，但它不可以擷取任何值。

下一範例程式定義 calculateSquare 函式（第 2 行），此函式的型態是回傳 Int 的型態的函式。calculateSquare 函式又包含 answer 函式（第 4 行）。巢狀的 answer 函式擷取兩個值分別是 n 與 square。擷取這些值後，calculateSquare 函式回傳 answer 函式，將此函式當做閉包，而此閉包是將 square 乘上 n 後，再指定給 square，最後將 square 回傳，所以可以說回傳 answer，其實就是將 square 乘以 n。

範例程式

```
01 // capturing values
02 func calculateSquare(forNumber n: Int) -> () -> Int {
03     var square = 1
04     func answer() -> Int {
05         square = square * n
06         return square
07     }
08     return answer
09 }
10
11 let squareByFive = calculateSquare(forNumber: 5)
12 print(squareByFive())
13 print(squareByFive())
14 print(squareByFive())
15 print(squareByFive())
```

輸出結果

```
5
25
125
625
```

calculateSquare 函式的回傳型態是 () -> Int。此表示它將回傳一函式，此回傳函式沒有參數，而且每一次呼叫時會回傳 Int 值。calculateSquare 函式定義整數變數 square，用以儲存目前 answer 函式的回傳值，其初始值為 1。第一次呼叫 squareByFive() 函式的結果是 5。第二次呼叫 squareByFive() 函式的結果是 25，因為此時的 square 值是 5，所以 5*5 是 25。第三次呼叫 squareByFive() 函式的結果是 125，因為此時的 square 值是 25，所以 25*5 是 125。依此類推，第四次呼叫 squareByFive() 函式的結果是 625。

8.4 閉包是參考型態

閉包是屬於參考型態 (reference type)。承上節的範例程式，將 squareByFive 指定給 alsosquareByFive，然後再呼叫 alsosquareByFive()將得到 3125。因為承上題的原故，所以得到的結果是 625*5。

範例程式

```
16 // closure as reference type
17 let alsosquareByFive = squareByFive
18 print(alsosquareByFive())
```

輸出結果

```
3125
```


自我練習題

1. 請問下列程式的輸出結果

(a)

```
func makeDecrementor(forDecrement amount: Int) -> () ->Int {
    var total = 100
    func Decrementor() ->Int {
        total -= amount
        return total
    }
    return Decrementor
}

let DecrementByTen = makeDecrementor(forDecrement: 10)
print(DecrementByTen())
print(DecrementByTen())
print(DecrementByTen())
print("")
let DecrementByEight = makeDecrementor(forDecrement: 8)
print(DecrementByEight())
print(DecrementByEight())
```

(b)

```
// trailing closure
let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]

var finished = numbers.sorted(by: {$0 > $1})

print("排序前資料: ")
for i in numbers {
    print("\(i) ", terminator: "")
}
print("")

print("排序後資料: ")
for i in finished {
    print("\(i) ", terminator: "")
}
print("")
```

2. 請將以下的程式加以除錯。

(a)

```
let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]

var finished = numbers.sort(((a: Int, b: Int) -> Bool in return a < b)))
for i in finished {
    print("\(i) ", terminator: "")
}
print("")
```

(b)

```
let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]

var finished = numbers.sort({a, b return a < b })

println("排序前資料: ")
for i in numbers {
    print("\(i) ", terminator: "")
}
print("")

println("排序後資料: ")
for i in finished {
    print("\(i) ", terminator: "")
}
print("")
```

(c)

```
let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]

var finished = numbers.sort({$a < $b})

printin("排序前資料: ")
for i in numbers {
    print("\(i) ", terminator: "")
}
print("")

println("排序後資料: ")
for i in finished {
    print("\(i) ", terminator: "")
```



```
}
print("")
```

(d)

```
let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]

var finished = numbers.sort() ((a: Int, b: Int) -> Bool in return a < b))
println("排序前資料: ")
for i in numbers {
    print("\(i) ", terminator: "")
}
print("")

println("排序後資料: ")
for i in finished {
    print("\(i) ", terminator: "")
}
print("")
```

(e)

```
// trailing closure
let numbers = [10, 8, 20, 7, 56, 3, 2, 1, 99]

var finished = numbers.sorted((0 < $1))

println("排序前資料: ")
for i in numbers {
    print("\(i) ", terminator: "")
}
print("")

println("排序後資料: ")
for i in finished {
    print("\(i) ", terminator: "")
}
print("")
```