(e)

```
var content2 = "Hello, Swift"

print(content2.count)
print(content2.dropFirst())
print(content2.dropFirst(3))
print(content2.dropLast())
print(content2.dropLast(3))
print(content2)
```


3 CHAPTER 運算子

運算子(operator)通常是一符號(symbol)，它具有特定的功能，如＋表示是一加法的符號。

學習運算子時，要注意的是運算子的運算優先順序(priority)及結合性(associative)。運算優先順序愈高表示要先運算，而結合性表示是由左至右運算，或是由右至左運算。大部份的運算子的結合性是由左至右，少數是由右至左。

常用的 Swift 運算子計有算術運算子、關係運算子、邏輯運算子、位元運算子、遞增與遞減運算子及指定運算子。茲分別敘述如下：

3.1 算術運算子

Swift 的算術運算子(arithmetic operator)計有 + (加)、- (減)、* (乘)、/ (除)、% (兩數相除取其餘數)。一般的算術的運算規則是「先乘除，後加減」。所以 *、/、% 的運算優先順序高於 +、-。算術運算子的結合性是由左至右，不過可利用小括號改變其運算的順序。請參閱以下範例程式。

範例程式

```
01 // arithmetic operator
02 let a = 100, b = 30
03 print("\(a) + \(b) = \(a+b)")
04 print("\(a) - \(b) = \(a-b)")
05 print("\(a) * \(b) = \(a*b)")
```

```
print("\(a) / \(b) = \(a/b)")
print("\(a) % \(b) = \(a%b)")

let d = Double(a) / Double(b)
print("\(a) / \(b) = \(d)")
```

輸出結果

```
100 + 30 = 130
100 - 30 = 70
100 * 30 = 3000
100 / 30 = 3
100 % 30 = 10
100 / 30 = 3.33333333333333
```

要注意的是，兩個整數相除，其結果是整數，如 100 / 30，答案是 3。

若要得到正確的答案可利用型態轉換(type casting)，如範例中的

```
Double(a) / Double(b)
```

暫時將 a 與 b 變數由整數型態轉為 Double 的資料型態，而敘述

```
100 % 30
```

表示 100 除以 30 的餘數是 10。注意，

```
a / Double(b)
```

是不行的，因為兩數相除時，兩數的資料型態要一樣才可以。

+ 運算子也可用於字串與字串或字元之間的連接，如下範例程式所示：

範例程式

```
let concateStr: String = "Hello " + "Swift"
print(concateStr)
let concateStrAndChar: String = "iPhone " + "6"
print(concateStrAndChar)
```

輸出結果

```
Hello Swift
iPhone 6
```


上述敘述說明將字串 “Hello “ 與字串 “Swift” 相連，然後指定給 concateStr。同理，將 “iPhone ” 與 “6” 相連，再將它指定給 concateStrAndChar。

3.2 關係運算子

Swift 的關係運算子(relational operator)計有 <(小於)、<=(小於等於)、>(大於) >=(大於等於)、==(等於)、!=(不等於)。關係運算子也可稱為比較運算子(comparative operator)。

關係運算子的運算優先順序低於算術運算子，這表示在一運算式中，若有算術運算子，則會優先被運算。同類的關係運算子中，< 、<=、 >、 >= 的運算順序高於 == 與 != 。而此類的運算子之結合性也是由左至右。

經由關係運算子的運算式，其最後的結果不是真，就是假。若為真，則輸出結果 true，否則，輸出結果 false。請看以下範例程式。

範例程式

```
// relational operator
let a = 100, b = 30
print("\(a) > \(b) = \(a > b)")
print("\(a) >= \(b) = \(a >= b)")
print("\(a) < \(b) = \(a < b)")
print("\(a) <= \(b) = \(a <= b)")
print("\(a) == \(b) = \(a == b)")
print("\(a) != \(b) = \(a != b)")
```

輸出結果

```
100 > 30 = true
100 >= 30 = true
100 < 30 = false
100 <= 30 = false
100 == 30 = false
100 != 30 = true
```

關係運算子敘述最後的結果不是 true 就是 false，而不像 C 或 Objective-C 的結果為 1 或是 0。

3.3 邏輯運算子

Swift 的邏輯運算子 (logical operator) 計有 &&(且)、||(或)、!(反)。基本上，邏輯運算子的運算優先順序比最後一章的位元運算子來得低，但比指定運算子來得高。結合性是由左至右。但！的運算子是例外，它與 3.4 節所談論的遞增與遞減運算子相同。其中 && 又高於 ||。詳細情形請參閱本章後面的表 3-4。

邏輯運算子的目的是將條件變為嚴格或寬鬆。若利用 &&，則會將條件變為嚴格，因為兩個條件皆為真才為真。如表 3-1 所示：

表 3-1　邏輯運算子 && 的真值表

條件式 1	條件式 2	條件式 1 && 條件式 2
真	真	真
真	假	假
假	真	假
假	假	假

若利用 ||，則會使條件變為寬鬆，因為只要有一條件為真就為真。如表 3-2 所示：

表 3-2　邏輯運算子 || 的真值表

條件式 1	條件式 2	條件式 1 || 條件式 2
真	真	真
真	假	真
假	真	真
假	假	假

而！的功能有點像豬羊變色，將真變為假，或是將假變為真。如表 3-3 所示：

表 3-3　邏輯運算子！的真值表

條件式	！條件式
真	假
假	真

在

```
條件式 1 && 條件式 2
```

中，若條件式 1 為假，則結果將為假，因此，不必再看條件式 2。

在

```
條件式 1 || 條件式 2
```

中，若條件式 1 為真，則結果將為真，因此，不必再看條件式 2。請參閱下一範例程式。這種不必再看條件式 2 的方法，可使程式執行更有效率。

範例程式

```
01  // logical  operator
02  let a = 100, b = 30
03  let andOper: Bool = a > 90 && b < 20
04  print("\(a) > 90 && \(b) < 20  = \(andOper)")
05
06  let orOper: Bool = a > 90 || b < 20
07  print("\(a) > 90 || \(b) < 20  = \(orOper)")
08
09  let notOper: Bool = !(a > 90)
10  print("!(\(a) > 90) = \(notOper)")
```

輸出結果

```
100 > 90 && 30 < 20  = false
100 > 90 || 30 < 20  = true
!(100 > 90) = false
```

第一個敘述為假，乃是因為 b < 20 為假，因為真且假為假。第二個敘述為真，乃是因為真或假為真。最後敘述為假，乃是因為 100 > 90 為真，採取反的邏輯運算子，結果將為 false。

3.4 指定運算子與算術指定運算子

指定運算子是最常用到的，要注意的是運算式的左邊一定要為變數，這樣才可以接受右邊的值。指定運算子是目前討論到運算子中運算優先順序最低的，其結合性是由右至左。

當算術運算子與指定運算子合在一起時，此運算子稱為算術指定運算子(arithmetic assignment operator)。我們也將它歸類在指定運算子中。若 op 表示某一算術運算子，則下一敘述

```
x op= 10;
```

等同於

```
x = x op 10;
```

請參閱下一範例程式。

範例程式

```
01  // arithmetic assignment operator
02  var num = 100
03  print("num = \(num)")
04
05  num += 2
06  print("\n 加2後")
07  print("num = \(num)")
08
09  num -= 2
10  print("\n 減2後")
11  print("num = \(num)")
12
13  num *= 2
14  print("\n 乘2後")
15  print("num = \(num)")
16
17  num /= 2
18  print("\n 除2後")
19  print("num = \(num)")
```


輸出結果

```
num = 100

加2後
num = 102

減2後
num = 100

乘2後
num = 200

除2後
num = 100
```

其中

```
num += 2;
```

表示

```
num = num + 2;
```

依此類推。注意，num 會隨著程式的執行而改變。

有關運算子的運算優先順序，建議先記大原則，再來看哪些是例外的運算子。由高至低分別為遞增與遞減運算子、算術運算子、關係運算子、邏輯運算子，最後是指定運算子和算術指定運算子。

至於結合性除了遞增與遞減運算子、指定運算子和算術指定運算子，！及 ~ 運算子是由右至左外，其餘都是由左至右執行運算的。

表 3-4 是本章所提及有關運算子的運算優先順序與結合性的資訊，愈上面的運算子，其運算順序愈高，所以是由上往下遞減之。

3.5 兩數對調

一般我們要將兩數對調，皆會借助另一變數加以完成。如下範例所示：

範例程式

```
01  var a = 100
02  var b = 200
03  print("a = \(a), b = \(b)")
04
05  let temp = a
06  a = b
07  b = temp
08  print("a = \(a), b = \(b)") var a = 100
```

輸出結果

```
a = 100, b = 200
a = 200, b = 100
```

程式中利用 temp 變數做為暫時存放的空間，從輸出結果可得知，a 和 b 確實相互對調了。

新版的 Swift 4 不需要暫時的變數就可以達成，如下範例所示：

範例程式

```
01  var a = 100
02  var b = 200
03  print("a = \(a), b = \(b)")
04
05  (a, b) = (b, a)
06  print("a = \(a), b = \(b)")
```

輸出結果

```
a = 100, b = 200
a = 200, b = 100
```

程式中以

```
(a, b) = (b, a)
```

來完成 a 和 b 兩數的對調

表 3-4 Swift 運算子的運算優先順序與結合性

運算子	結合性
!	由右至左
* / %	由左至右
+ -	由左至右
< <= > >=	由左至右
== !=	由左至右
&&	由左至右
||	由左至右
= += -= *= /= %=	由右至左

除了上述的運算子外，還有位元運算子，由於它牽涉到運算子函式，所我們將於第 18 章再來討論。

自我練習題

1. 試問下列程式的輸出結果：

(a)

```
// arithmetic operator
let a = 200, b = 3
print("\(a) + \(b) = \(a+b)")
print("\(a) - \(b) = \(a-b)")
print("\(a) * \(b) = \(a*b)")
print("\(a) / \(b) = \(a/b)")
print("\(a) % \(b) = \(a%b)")

let d = Double(a) / Double(b)
print("\(a) / \(b) = \(d)")
```

(b)

```
// arithmetic operator
var a, b: Int
a = 80 + 60 * 3 - 20 / 2
b = 60 * 2 + 30 / 2 + 65

print("a = \(a)")
print("b = \(b)")

print("10 / 3 = \(10/3)")
print("10.0 / 3 = \(10.0/3)")
```

(c)

```
// relational operator
let a = 30, b = 100
print("\(a) > \(b) = \(a > b)")
print("\(a) >= \(b) = \(a >= b)")
print("\(a) < \(b) = \(a < b)")
print("\(a) <= \(b) = \(a <= b)")
print("\(a) == \(b) = \(a == b)")
print("\(a) != \(b) = \(a != b)")
```


(d)

```
// logical  operator
let a = 30, b = 100
let andOper: Bool = a > 90 && b < 20
print("\(a) > 90 && \(b) < 90  = \(andOper)")

let orOper: Bool = a > 90 || b < 20
print("\(a) > 90 || \(b) < 90  = \(orOper)")

let notOper: Bool = !(a > 90)
print("!(\(a) > 90) = \(notOper)")
```

(e)

```
// arithmetic assignment operator
var num = 20
print("num = \(num)")

num += 2
print("\n 加 2 後")
print("num = \(num)")

num -= 2
print("\n 減 2 後")
print("num = \(num)")

num *= 2
print("\n 乘 2 後")
print("num = \(num)")

num /= 2
print("\n 除 2 後")
print("num = \(num)")
```

(f)

```
var x = "94 強"
var y = "87 分"
print("x = \(x), y = \(y)")

let z = x
x = y
y = z
print("x = \(x), y = \(y)")

(x, y) = (y, x)
print("x = \(x), y = \(y)")
```


4

CHAPTER

迴圈敘述

在我們日常生活中，常常會將某些相同的事情處理多次。這也對應了 Swift 的迴圈敘述(loop statement)。在程式設計中，迴圈敘述就是重複執行某一些敘述。

Swift 的迴圈敘述計有 while、repeat...while，以及 for-in 等三種。我們將一一舉例說明之。

4.1 while 迴圈敘述

while 迴圈敘述的語法如下：

```
初值設定運算式
while 條件運算式 {
        迴圈主體敘述
        更新運算式
}
```

while 迴圈敘述要先判斷條件運算式是否為真，若是，則執行迴圈主體敘述與更新運算式，否則結束迴圈。

基本上，這三種迴圈敘述是可以交換使用的。因此，我們將以一些相同的題目，而以不同的迴圈敘述加以撰寫之。