基本語法（syntax）

準備好對你的軟體渦輪加速了嗎？想要有個**簡單**又**快速編譯**的程式語言嗎？不只**執行得很快**，還**易於發布**你的作品給使用者？那麼你已經**準備好 Go** 啦！

Go 是一種專注於**簡單**與**速度**的程式語言。它比其他程式語言都來得簡單所以很好學習。而且它也讓你可以直接運用最流行的多核心處理功能，來加速你的程式。這一章會走過所有讓**你的開發人生更簡單**的 Go 的功能，也讓你的**使用者更開心**。

www.gotop.com.tw

預備備、就定位、Go ！

時間回到 2007 年，Google 的搜尋引擎遇到了個麻煩。他們得維護擁有上百萬行原始碼的程式。在他們嘗試新的改變之前，得先把原始碼編譯成可執行的狀態，而運氣好的話可能至少得花上一個小時。不得不說，這對開發人員的產能是很糟糕的。

於是 Google 的工程師 Robert Griesemer、Rob Pike 以及 Ken Thompson 規劃出一個新的程式語言需要達到的目標：

- 快速編譯
- 更少繁瑣的程式碼
- 自動釋放閒置的記憶體空間（垃圾回收）
- 輕鬆寫出可以同步執行不同工作的軟體（並發）
- 對多核心處理器支援度高

經過多年的努力，Google 終於發明了 Go：一個可以快速編寫、快速編譯以及執行的程式語言。這個專案爾後在 2009 年轉授權為開放原始碼。現在大家都可以免費取用，而你正應該使用它！Go 很快地因為它的易於上手以及強大變得知名。

假如你編寫的是命令列工具，Go 可以從同一個程式碼，建立成 Windows、macOS 以及 Linux 可用的執行檔。假如你編寫的是網路伺服器，它可以幫助你搞定多使用者同時連線。除此之外，無論你寫什麼，它會幫你確認你的程式碼是否更加地易於維護以及成長。

等不及要開始了嗎？讓我們 Go 吧！

碁峰 www.gotop.com.tw

Go 的遊樂場

嘗試 Go 的最簡單方法是透過你的網頁瀏覽器造訪 *https://play.golang.org*。Go 團隊在這裡已經建立好簡單的編輯器環境讓你可以輸入 Go 程式碼，並且在他們的伺服器上執行。結果會直接呈現在你的瀏覽器上。

（當然只有在你網路順暢時才行得通。假如事與願違，前往第 25 頁學習如何在你的電腦上下載以及執行 Go 編譯器。接著透過你剛安裝好的編譯器執行下方的範例。）

讓我們現在就來試試看！

動手做！

1. 在你的瀏覽器上打開 *https://play.golang.org* 網址。（如果你看到的結果跟我們的截圖看起來不太一樣，別太擔心；這應該只是因為它們的網站在我們出版本書之後又更進步了！）
2. 刪除在編輯區的所有程式碼，並且輸入如下：

```
package main

import "fmt"

func main() {
     fmt.Println("Hello, Go!")
}
```

別擔心，我們晚點會在下一頁解釋這些是什麼意思！

3. 點擊 Format 按鈕，它會根據 Go 的語法慣例，自動重新格式化你的程式碼。
4. 點擊 Run 按鈕。

你應該會在螢幕的底部看到「Hello, Go!」。恭喜你，你第一次成功地執行了 Go 程式囉！

請翻到下一頁，我們即將解釋剛剛到底做了什麼…

輸出 → Hello, Go!

www.gotop.com.tw

這代表什麼意思？

你剛完成了你的第一個 Go 程式！現在讓我們仔細來看看程式碼並且搞清楚它們真正的涵義…

任何一份 Go 檔案都會以 package 子句作為開始。**套件**是做類似事情的程式碼的集合，比如說字串格式化或者繪圖。Package 子句提供了該套件一個名字，整份文件中的程式碼都會歸屬在這個套件名下。現在我們使用了特殊的套件名稱 main，假如程式碼需要被直接執行（通常是在終端機），那麼就得使用這個特殊的套件名稱。

接下來 Go 通常都會有至少一個 import 陳述句。每個檔案在使用其他套件擁有的程式碼之前，必須**匯入（import）**這些套件。同時讀取電腦內所有的 Go 程式碼會產生一個笨重又緩慢的程式，不如只針對特定你所需要的套件引入即可。

```
package main

import "fmt"

func main() {
        fmt.Println("Hello, Go!")
}
```

這一行說該份檔案中所有的程式碼都歸屬於「main」套件。

這裡說我們即將從「fmt」套件使用可讓文字格式化的程式碼。

「main」函式是特別的；當你的程式啟動時它會第一個被執行。

這裡會調用「fmt」套件的函式「Println」。

這一行會在你的終端機（或者網頁瀏覽器，假如你是用 Go 遊樂場）顯示（「印出」）「Hello, Go!」。

在 Go 檔案中的最後才是主要的程式碼，通常會分成好幾個函式（function）。**函式（function）**是一至多行程式碼的組合，你可以在你的程式中任何一處**調用（call）**（執行）它們。Go 的程式在開始執行時，會先尋找名為 main 的函式並且第一個執行它，這也正是為何我們要把這段函式命名為 main。

Fun 輕鬆

當你還無法馬上參透一切，別擔心！

我們馬上會在下面幾頁講得更詳細些。

典型的 Go 檔案佈局

你很快就會習慣在大部分碰到的 Go 檔案中，看到以下三個段落的排列方式：

1. 套件子句
2. 任何 import 陳述句
3. 主要的程式碼

```
套件子句 {package main

匯入段落 {import "fmt"

主要程式碼 {func main() {
                fmt.Println("Hello, Go!")
           }
```

有句話是這麼說的：「當所有東西都好好歸位，那麼整潔就是件自然而然的事情」。Go 是一個相當一致的程式語言。這是個相當好的習慣：當你拿到一份 Go 的程式碼，你自然會知道要如何在整個專案中找到你要的。

碁峰 www.gotop.com.tw

問：**我其他的程式語言需要在每一行的陳述句最後加上分號。為何 Go 不用？**

答：你當然可以在 Go 使用分號來分別每一個陳述句，但不受強制（事實上，通常是不允許的）。

問：Format **按鈕是做什麼用的？為何我們要在執行程式碼之前按下它呢？**

答：Go 編譯器有一個叫做 `go fmt` 的標準格式化工具。這個 Format 按鈕就是網頁版的 `go fmt`。

無論你何時分享程式碼，其他的 Go 開發者會預期你的程式碼是遵循標準 Go 格式的。像是縮排以及空格會被格式化成標準的結果，來讓所有人都能輕鬆的閱讀。在其他的語言都倚靠使用者，手動地重新格式化他們的程式碼來符合風格規範時，你在 Go 只需要執行 `go fmt` 自動地達到這個目標。

本書中我們所建立的所有範例都已經執行過這個格式化工具，而你也應該讓你自己的程式碼做到這件事！

假如發生錯誤呢？

Go 程式需要遵守特定的規則來避免讓編譯器困惑。假如我們破壞了任何其中一條規則，錯誤訊息就會隨之而來。

假設我們忘記在第六行調用 `Println` 時加上括號。

假如我們打算執行這個版本的程式，就會得到錯誤：

假設我們忘了這邊該要有的括號…

```
行數 1 package main
     2
     3 import "fmt"
     4
     5 func main() {
     6         fmt.Println "Hello, Go!"
     7 }
```

Go 遊樂場所使用的檔案名稱

發生錯誤的行數

錯誤的解釋

```
prog.go:6:14: syntax error: unexpected literal "Hello, Go!" at end of statement
```

發生錯誤時的行數以及字元數

Go 會告訴我們需要到程式碼檔案中的第幾行來修正問題（Go 遊樂場會在執行之前，把你的程式碼先存到一個叫做 *prog.go* 的暫時檔）。接著會告訴你這個錯誤的描述。在這個案例，由於我們刪除了括號，Go 無從得知我們是否真的要調用 `Println` 這個函式，於是它無法理解為何我們要把 `"Hello, Go"` 放在第六行的最後面。

www.gotop.com.tw

拆解東西真的是很有教育性！

透過不同的方法拆解程式，可以讓我們對於 Go 程式的規則有更深一層的了解。以這個程式碼例子為基礎，嘗試用下方任一種方法改變內容後並執行看看。然後恢復你做過的改變，接著嘗試下一組。看看會發生什麼事情！

```
package main

import "fmt"

func main() {
        fmt.Println("Hello, Go!")
}
```

嘗試拆解程式碼範例並看看會發生什麼事！

假如你這麼做…	…它會因為…而發生錯誤
刪除套件子句 ~~package main~~	每個 Go 檔案必須以套件子句作為開頭。
刪除 import 陳述句… ~~import "fmt"~~	每個 Go 檔案必須透過匯入的方式引用套件。
匯入第二個（沒用到的）套件… import "fmt" import "strings"	Go 必須只匯入有在程式碼當中引用的套件（這讓你的程式碼編譯得更快！）
重新命名為 main 函式… func ~~main~~hello	Go 會優先執行名為 main 的函式。
把 Println 改為小寫後調用… fmt.Pprintln("Hello, Go!")	Go 是有區分大小寫的，縱使 fmt.Println 可用，並不代表 fmt.println 也可行。
把 Println 之前的套件名稱刪除…. ~~fmt.~~Println("Hello, Go!")	Println 函式並不是 main 套件的一部分，於是 Go 調用該函式時需要它的套件名稱。

讓我們先試試看第一個範例…

刪除 package 子句…

```
import "fmt"

func main() {
        fmt.Println("Hello, Go!")
}
```

你會得到這樣的錯誤！

```
can't load package: package main:
prog.go:1:1: expected 'package', found 'import'
```


碁峰 www.gotop.com.tw

調用函式

我們的範例中調用了 fmt 的 Println 函式。為了調用函式，首先輸入函式名稱（在這個案例是 Println），接著是一對括號。

```
package main

import "fmt"

func main() {
        fmt.Println("Hello, Go!")
}
```

調用 Println 函式

我們很快就會解釋這塊！

函式名稱

```
fmt.Println()
```

括號

像不少函式一樣，Println 可以擁有一至多個**引數**：你打算在函式內使用的值。引數會出現在函式名稱後面的括號內。

括號內有一至多個引數，以逗號分開。

```
fmt.Println("First argument", "Second argument")
```

輸出 →

```
First argument Second argument
```

Println 在調用時可以不需要引數，也可以有很多引數。當我們在之後遇到更多函式，我們會發現大部分的情境是調用固定數量的引數。假如你調用了太少或太多，你會得到錯誤訊息，說預期要有幾個引數，而你必須去解決這個問題。

Println 函式

在你打算知道你的程式在做什麼時可以使用 Println 函式。任何你傳遞的引數都會在終端機畫面，以一個逗號以及空格的間隔印出來（顯示出來）。

在印出所有的引數之後，Println 會在終端機畫面跳到下一行（這正是為什麼這個函式最後以「ln」作為結尾）。

```
fmt.Println("First argument", "Second argument")
fmt.Println("Another line")
```

輸出 →

```
First argument Second argument
Another line
```


www.gotop.com.tw

從其他套件（packages）使用函式

在我們的第一個程式當中，所有程式碼都是 main 套件的一部分，然而 Println 卻是屬於 fmt 套件（fmt 是「format」的意思）。為了能夠調用 Println 函式，我們得先匯入擁有該函式的套件。

```
package main

import "fmt"

func main() {
        fmt.Println("Hello, Go!")
}
```

在可以存取 Println 之前，我們得引入「fmt」套件。

這裡指出我們調用的函式是屬於「fmt」套件。

一旦我們匯入了該套件，我們可以存取該套件中任何函式，只要輸入套件的名稱、接著是一個句點、最後才是我們要使用的函式名稱。

套件名稱　　函式名稱

```
fmt.Println()
```

以下是我們如果要從其他套件調用函式的程式碼範例。由於我們需要匯入多個套件，我們改用另一種 import 陳述句的語法，來讓你可以透過括號，以及一個套件名稱一行的方式匯入多個套件。

```
package main

import (
        "math"
        "strings"
)

func main() {
        math.Floor(2.75)
        strings.Title("head first go")
}
```

另一種「import」陳述句的語法，讓你可以一次匯入多個套件。

匯入「math」套件，讓我們可以使用 math.Floor。

匯入「strings」套件，讓我們可以使用 strings.Title。

從「math」套件調用 Floor 函式。

從「strings」套件調用 Title函式。

這個程式並沒有輸出。（我們很快就會解釋為什麼！）

一旦我們匯入了 math 以及 strings 套件，我們就可以透過 math.Floor 存取 math 套件的 Floor 函式，以及透過 strings.Title 存取 strings 套件的 Title 函式。

你可能已經發現在這個程式碼中，除了調用了兩次函式之外，這個範例並沒有輸出任何結果。接著我們會看看如何修正。

碁峰 www.gotop.com.tw

函式回傳值

在上一頁的程式碼範例，我們嘗試調用 math.Floor 以及 strings.Title 函式，然而它們並不會產生任何輸出：

```
package main

import (
        "math"
        "strings"
)

func main() {
        math.Floor(2.75)
        strings.Title("head first go")
}
```

這個程式沒有輸出結果。

當我們調用 fmt.Println 函式，我們不需要額外地在事後多做溝通。我們會傳遞一至多個值給予 Println 來印出，並且相信會確實印出結果。然而有時候程式會需要在調用函式之後可以獲得回傳的值。因此大部分程式語言的函式擁有**回傳值**：一種調用的函式運算完之後回傳給調用者的值。

math.Floor 以及 strings.Title 函式都可作為函式回傳值的範例。math.Floor 在取得浮點數之後，向下捨去小數後取得最接近的整數，並且回傳這個整數的結果。而 strings.Title 函式取得一個字串後，將每一個單字的首字大寫（也就是轉換成「首字大寫格式」），接著回傳這個首字大寫化後的字串。

若要實地看看這些函式調用後的結果，我們需要把這些回傳值傳遞給 fmt.Println：

```
package main

import (
        "fmt"
        "math"
        "strings"
)

func main() {
        fmt.Println(math.Floor(2.75))
        fmt.Println(strings.Title("head first go"))
}
```

一樣匯入「fmt」套件。

把 math.Floor 回傳的結果透過調用 fmt.Println 印出來。

取得數字，無條件捨去後接著回傳該結果。

輸出

```
2
Head First Go
```

把 strings.Title 回傳的結果透過調用 fmt.Println 印出來。

取得字串，並且回傳已經將每個單字的首字大寫處理過的新字串。

修改後的結果會回傳，並且印出來，於是我們就可以直接看到結果。

碁峰 www.gotop.com.tw

池畔風光

你的**工作**是把游泳池內的程式碼片段，放到上方程式碼中空白的地方。同一個程式碼片段**不能**使用超過一次，而且也不需要把游泳池內所有的片段都用完。你的**目標**是讓這整段程式碼可以正常運作，並且產生所列出的輸出結果。

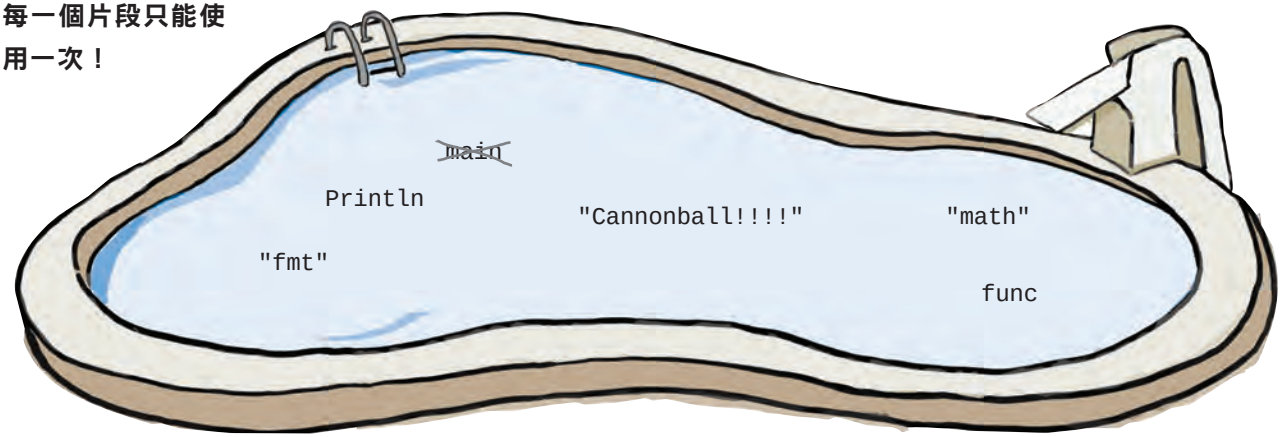
```
package main
import (
        ____
)
____ main() {
        fmt.Println(_________________)
}
```

我們已幫你完成了這塊！

輸出

```
Cannonball!!!!
```

注意：游泳池內的每一個片段只能使用一次！



答案在第 29 頁。

碁峰 www.gotop.com.tw

Go 程式樣板

在這邊的程式碼片段，想像一下把它們放進完整的 Go 程式內：

如果可以的話，把整段程式先輸入到 Go 遊樂場，並且輸入你自己的片段，看看會發生什麼結果。

```
package main

import "fmt"

func main() {
        fmt.Println(                    )
}
```

在這裡輸入你的程式碼！

字串（strings）

我們已經把**字串**（**strings**）當作引數傳遞給 `Println`。字串是一系列代表著字元的位元組。你可以使用**字串字面量**（**string literals**）的方式來直接定義字串：也就是在雙引號之間擺放文字，Go 就會視為字串。

雙引號開頭 `"Hello, Go!"` 雙引號結尾

輸出

```
Hello, Go!
```

透過字串，像是換行、跳位（tab）或者其他不易在程式碼中引用的字元，可以透過**跳脫序列**（**escape sequences**）來呈現：也就是反斜線後面接著代表其他字元的特殊字元。

跳脫序列	值
\n	換行字元
\t	跳位字元
\"	雙引號
\\	反斜線

在字串內換行　　輸出

`"Hello,\nGo!"`	`Hello,` `Go!`
`"Hello,\tGo!"`	`Hello,  Go!`
`"Quotes: \"\""`	`Quotes: ""`
`"Backslash: \\"`	`Backslash: \`

www.gotop.com.tw

符文（runes）

```
package main
import "fmt"

func main() {
        fmt.Println(                    )
}
```

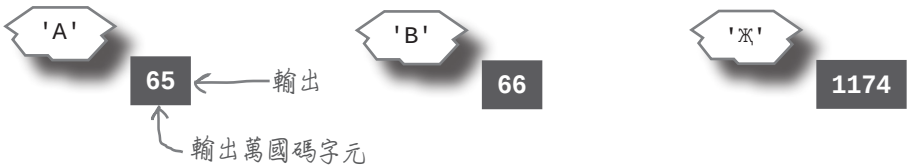
我們的樣本又來了…

在這裡輸入你的程式碼！

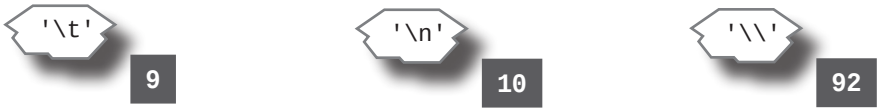
當字串通常用來呈現完整的字元序列時，Go 的**符文**（**runes**）卻是被用作呈現單一字元。

字串字面量用雙引號（"）圍繞來編寫，而**符文字面量**（**runes literals**）卻是用單引號（'）來編寫。

Go 程式幾乎可以使用世界上任何語言的任何字元，因為 Go 使用萬國碼（Unicode）標準來儲存符文。符文被當作數值而不是字元本身來儲存，假如你傳遞一組符文給 `fmt.Println`，你會在輸出看見數值而不是原始的字元。



就像是字串字面量，跳脫序列也可以在字符字面量用來呈現難以在程式碼中引用的字元。



布林值（Booleans）

布林（**Boolean**）值只有兩種值：`true` 或者 `false`。特別是在條件陳述句中相當實用，會讓條件為真或假時，只讓部分程式碼被執行（我們會在下一章的條件式中解釋）。

碁峰 www.gotop.com.tw

數值（numbers）

你也可以直接用程式碼定義數值，而這比字串字面量還要簡單：只需要輸入數字即可。

```
package main

import "fmt"

func main() {
        fmt.Println(                    )
}
```

我們的樣本又來了…

在這裡輸入你的程式碼！

```
42
```

整數

```
42
```

輸出

```
3.1415
```

浮點數

```
3.1415
```

我們很快地帶過， Go 把整數以及浮點數視為不同的型別，所以記住小數點可作為識別整數以及浮點數的方法。

數學運算以及比較

Go 的基本數學運算與其他大部分程式語言一樣。+ 符號作為加法，- 作為減法，* 作為乘法，而 / 作為除法。

```
1 + 2
```

```
3
```

```
5.4 - 2.2
```

```
3.2
```

```
3 * 4
```

```
12
```

```
7.5 / 5
```

```
1.5
```

你可以使用 < 以及 > 來比較兩個數值的大小。你也可以使用 ==（這是兩個等號）來看兩個值是否相等，而 !=（這是驚嘆號以及等號，念成「不等於」）來看看兩個值是否不相等。<= 測試是否第一個值小於或等於第二個值，而 >= 測試是否第一個值大於或等於第二個值。

比較的結果是一個布林值，也就是 `true` 或者 `false`。

```
4 < 6
```

```
true
```

```
4 > 6
```

```
false
```

```
2+2 == 5
```

```
false
```

```
2+2 != 5
```

```
true
```

```
4 <= 6
```

```
true
```

```
4 >= 4
```

```
true
```


碁峰 www.gotop.com.tw

型別（types）

在剛剛的程式碼範例，我們看到了 math.Floor 函式把浮點數無條件捨去到最接近的整數，而 strings.Title 函式轉換字串為首字大寫的格式。把數值作為引數傳遞給 Floor 函式，以及把字串作為引數傳遞給 Title 是很合理的。但是假如我們傳遞了字串給 Floor、而數值傳遞給 Title，會發生什麼事呢？

```
package main

import (
        "fmt"
        "math"
        "strings"
)

func main() {
        fmt.Println(math.Floor("head first go"))
        fmt.Println(strings.Title(2.75))
}
```

通常得使用浮點數！

通常得使用字串！

錯誤

```
cannot use "head first go" (type string) as type float64 in argument to math.Floor
cannot use 2.75 (type float64) as type string in argument to strings.Title
```

Go 針對各自的函式調用，印出了兩個錯誤訊息，而程式根本沒有執行！

這個世界上在你周遭的環境中，事物通常可以依據它們的用途，定義出不同的型別：你不會吃一台車或卡車當早餐（因為它們是車輛），而你也不會駕駛歐姆蛋或是一碗燕麥片去上班（因為它們是早餐）。

同樣地，Go 的值全部都已經被定義為不同的**型別**（**types**），這些型別明確地定義出數值的用途。整數可被用作數值運算，但是字串不行。字串可以被大寫化，但是數值無法，諸如此類。

Go 屬於**靜態型別**（**statically typed**），意味著 Go 在程式運作之前就清楚知道你使用的值的型別。函式預期它們的引數是特定的型別，而它們所回傳的值也擁有型別（可能與引數的型別相同，也有可能不同）。假如你不小心在錯誤的地方用了錯誤的型別，Go 會透過錯誤訊息提醒你。好處是：這讓你在使用之前就能夠找到問題所在！

Go 是靜態型別的語言。假如你在錯誤的地方使用了錯誤的型別，Go 會讓你知道的。

碁峰 www.gotop.com.tw

型別（types）（續）

你可以透過傳遞值給 reflect 套件的 TypeOf 函式來得知它們的型別。讓我們來看看我們之前看過的值的型別吧：

```
package main

import (
        "fmt"
        "reflect"
)

func main() {
        fmt.Println(reflect.TypeOf(42))
        fmt.Println(reflect.TypeOf(3.1415))
        fmt.Println(reflect.TypeOf(true))
        fmt.Println(reflect.TypeOf("Hello, Go!"))
}
```

匯入「reflect」套件這樣我們才可使用 TypeOf 函式。

回傳引數的型別

輸出

```
int
float64
bool
string
```

這裡是以上的型別的用途：

型別	描述
int	也就是 integer，擁有的是整數。
float64	也就是 floating-point 數（浮點數），即有小數點的數值（之所以 64 會在型別名稱中，是因為 64 位元的資料都用作儲存這個數值。也就是說 float64 的值可以相當地、卻不是無限地精準到最後一位）。
bool	布林值。只能是 true 或者 false。
string	字串。用來呈現一系列字元的資料結構。

連連看，把每一個程式碼片段連到對應的型別。有些型別會有不只一個程式碼片段對應到它們。

reflect.TypeOf(25)	int
reflect.TypeOf(true)	
reflect.TypeOf(5.2)	float64
reflect.TypeOf(1)	
reflect.TypeOf(false)	bool
reflect.TypeOf(1.0)	
reflect.TypeOf("hello")	string

答案在第 29 頁。

答案
www.gotop.com.tw

宣告變數（variables）

變數在 Go 是一種儲存值的方式。你可以透過**宣告變數**（**variable declaration**）給予變數一個名稱。只要使用 var 鍵詞接著是你預期的變數名稱即可。

「var」鍵詞　變數名稱　型別

var quantity int

變數名稱　變數所持有的型別

```
var quantity int
var length, width float64
var customerName string
```

你可以同時宣告一樣型別的多個變數。

當你宣告了一個變數，你可以透過 =（只有一個等號）指派該型別的任何值給予這個變數：

```
quantity = 2
customerName = "Damon Cole"
```

你可以在同一個陳述句宣告一樣的值給多個變數。只要把多個變數名稱放在 = 的左邊，而同樣數量的值放在右邊，並且以逗號區隔即可。

```
length, width = 1.2, 2.4
```

同時指派多個變數。

一旦你給變數指派值了，你可以在程式碼內的任何一處，當你想要使用原始那個值的時候使用該變數：

```
package main

import "fmt"

func main() {
    var quantity int
    var length, width float64
    var customerName string

    quantity = 4
    length, width = 1.2, 2.4
    customerName = "Damon Cole"

    fmt.Println(customerName)
    fmt.Println("has ordered", quantity, "sheets")
    fmt.Println("each with an area of")
    fmt.Println(length*width, "square meters")
}
```

宣告變數

指派值給變數

使用變數

```
Damon Cole
has ordered 4 sheets
each with an area of
2.88 square meters
```


碁峰 www.gotop.com.tw

宣告變數（variables）（續）

假如你早就知道變數的值是什麼，你可以在同一行定義變數以及指派它們的值：

只需要在最後面加上指派。

宣告以及指派變數

```
var quantity int = 4
var length, width float64 = 1.2, 2.4
var customerName string = "Damon Cole"
```

假如你宣告了多個變數，也需要提供同樣數量的值。

你可以指派新的值給予現存的變數，不過它們得是同樣的型別。Go 的靜態型別保證你不會不小心指派錯誤型別的值給變數。

指派型別與宣告型別不同時

```
quantity = "Damon Cole"
customerName = 4
```

錯誤

```
cannot use "Damon Cole" (type string) as type int in assignment
cannot use 4 (type int) as type string in assignment
```

假如同時在宣告時指派了值給變數，你通常可以在宣告的部分省略變數型別的宣告。指派值的型別即會成為該變數的型別。

省略變數型別

```
var quantity = 4
var length, width = 1.2, 2.4
var customerName = "Damon Cole"
fmt.Println(reflect.TypeOf(quantity))
fmt.Println(reflect.TypeOf(length))
fmt.Println(reflect.TypeOf(width))
fmt.Println(reflect.TypeOf(customerName))
```

```
int
float64
float64
string
```

零值

假如你在宣告變數時沒有指派值，變數就會持有這個型別的**零值**（**zero value**）。數值型別的零值是 0：

```
var myInt int
var myFloat float64
fmt.Println(myInt, myFloat)
```

「int」型別的零值是 0。

```
0 0
```

「float64」型別的零值是 0。

然而對其他型別來說，0 這個值是不可行的，所以其他型別的零值就會是不同的東西。比如說 string 的零值是空字串，而 bool 的零值為 false。

```
var myString string
var myBool bool
fmt.Println(myString, myBool)
```

「string」的零值是空字串。

```
false
```

「bool」的零值是 false。

碁峰 www.gotop.com.tw

程式碼磁貼

有個 Go 程式在冰箱上被搞得亂七八糟。你是否可以重組程式碼磁貼來製作一個可執行的程式，以產生所求的輸出嗎？

輸出

```
I started with 10 apples.
Some jerk ate 4 apples.
There are 6 apples left.
```

```
, "apples.")
, "apples.")
, "apples left.")
var
var
int
originalCount
func main() {
}
int
originalCount
fmt.Println("I started with",
eatenCount
fmt.Println("Some jerk ate",
=
=
fmt.Println("There are",
10
4
eatenCount
package main
originalCount-eatenCount
import (
"fmt"
)
```

答案在第 30 頁。

碁峰 www.gotop.com.tw

短變數宣告（short variable declarations）

我們有說過你可以在同一行宣告以及指派值變數。

只需要在宣告的最後面進行指派。

宣告以及指派值給變數

```
var quantity int = 4
var length, width float64 = 1.2, 2.4
var customerName string = "Damon Cole"
```

假如你宣告了多個變數，就得提供一樣數量的指派值。

然而當某個變數要被宣告，而你假如早就知道它該被指派的值時，我們通常比較會使用**短變數宣告**（**short variable declaration**）。與其明確地定義變數的型別以及後面透過 = 指派值，你可以透過 := 一次達成。

讓我們來使用短變數宣告更新之前的範例吧：

宣告以及指派值給變數

```
package main

import "fmt"

func main() {
    quantity := 4
    length, width := 1.2, 2.4
    customerName := "Damon Cole"

    fmt.Println(customerName)
    fmt.Println("has ordered", quantity, "sheets")
    fmt.Println("each with an area of")
    fmt.Println(length*width, "square meters")
}
```

```
Damon Cole
has ordered 4 sheets
each with an area of
2.88 square meters
```

我們並不需要明確地宣告變數的型別；指派給變數的型別馬上就會變成變數自己的型別。

由於短變數宣告實在是相當地方便與簡潔，比起正規的宣告變數它們更常被廣泛地使用。你仍然會偶爾看見兩種不同的宣告方式，所以對兩種做法都駕輕就熟是相當重要的。

www.gotop.com.tw

拆解東西真的是很有教育性！

看到下面這個使用變數的程式，嘗試用下方任一種方法改變內容後並執行看看。然後恢復你做過的改變，接著嘗試下一組。看看會發生什麼事情！

```
package main

import "fmt"

func main() {
        quantity := 4
        length, width := 1.2, 2.4
        customerName := "Damon Cole"

        fmt.Println(customerName)
        fmt.Println("has ordered", quantity, "sheets")
        fmt.Println("each with an area of")
        fmt.Println(length*width, "square meters")
}
```

```
Damon Cole
has ordered 4 sheets
each with an area of
2.88 square meters
```

假如你這麼做…		…它會因為…而發生錯誤
對同一個變數進行第二次宣告。	`quantity := 4` `quantity := 4`	同一個變數只能宣告一次（縱使你可以隨時指派新的值給它，你也可以在別的範圍宣告同一個變數名稱。我們會在下一章學到什麼是範圍）。
刪除短變數宣告的：	`quantity = 4`	假如你忘了：，它會被當作指派而不是宣告，而你不能對還沒宣告的變數指派值。
把 string 的值指派給 int 的變數	`quantity := 4` `quantity = "a"`	變數只能指派同一個型別的值。
變數與值的數量不同	`length, width := 1.2`	你得提供每一個變數相對應的值，反過來說每一個值都得對應到一個變數。
移除有用到變數的程式碼	~~`fmt.Println(customerName)`~~	所有的變數都應該在你的程式中被使用到。假如你移除了用到變數的程式碼，你也得移除該變數的宣告。

碁峰 www.gotop.com.tw