# 附錄 延伸習題

／本章提要／

以下習題的解答, 都可在本書可下載的範例程式中找到 (比如延伸習題 01.01 的解答會位於資料夾 F1741\Chapter\Activity01.01 下), 請各位自行參閱。

## 延伸習題 01.01：定義與列印

這個延伸習題的目的，是要為醫師建立一個醫療表格，以便填寫患者的姓名、年齡、以及是否對花生過敏等等：

1. 首先建立以下變數：
   - 名字為字串
   - 姓氏也是字串
   - 年齡是整數 int
   - 過敏與否以布林值 bool 表示
2. 確保以上變數都已賦予初始值
3. 將資料值印到主控台畫面上

預期的輸出應該像這樣：

**執行結果**

```
Bob
Smith
34
false
```

## 延伸習題 01.02：指標值替換

在這個延伸習題中，你的工作是補完一位同事留下的工作。你的任務是要補足缺漏的程式碼，用註解提供說明，而且必須讓變數 a 和 b 的值交換。swap() 函式只能接收指標，而且不會傳回任何值：

```
package main

import "fmt"
```

```
func main() {
    a, b := 5, 10
    // 在此呼叫交換函式
    fmt.Println(a == 10, b == 5)
}

func swap(a *int, b *int) {
    // 在此交換變數值
}
```

以下是應該要看到的輸出：

**執行結果**

```
true true
```

## 延伸習題 01.03：訊息錯誤

以下的程式碼出了錯，原作者也束手無策，因此向你求助。你能讓它運作起來嗎？

```
package main

import "fmt"

func main() {
    count := 5
    if count > 5 {
        message := "Greater than 5"
    } else {
        message := "Not greater than 5"
    }
    fmt.Println(message)
}
```

提示：問題在變數 message 的定義位置。先執行程式觀察來錯誤訊息，然後修改程式，它應該能顯示以下輸出：

**執行結果**

```
Not greater than 5
```

## 延伸習題 01.04：計數錯誤

闖禍的朋友又來找你了，這回他又遇上一個程式臭蟲。程式碼應該輸出結果為 true, 但總是輸出 false。你能修正他犯的錯嗎？

```
package main

import "fmt"

func main() {
    count := 0
    if count < 5 {
        count := 10
        count++
    }
    fmt.Println(count == 11)
}
```

提示：問題在 count 變數的定義。修改程式後，它應該能顯示以下輸出：

**執行結果**

```
True
```

## 延伸習題 02.01：實作 FizzBuzz

在應徵某個程式設計工作時，面試官當面考驗你：你要寫出一段程式，其中有必須遵守的規則。問題大致如下：

1. 寫一段可以從 1 顯示到 100 的程式。

2. 如果顯示的數字是 3 的倍數，改為顯示文字『Fizz』。

3. 如果顯示的數字是 5 的倍數，改為顯示『Buzz』。

4. 如果顯示的數字是 3 和 5 的公倍數，改為顯示『FizzBuzz』。

提示：

1. 你可以用 strconv.Itoa() 函式將數字轉換成字串。

2. 寫一個可以迭代 100 次的迴圈，用一個變數來追蹤檢查的數字。第一個數字是 1，最後一個是 100。使用上述的計數變數和餘數除法算符 %，檢查是否為 3 或 5 的倍數。

輸出應該像下面這樣：

**執行結果**

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
…(以下略)
```

## 延伸習題 02:02：用 range 來走訪 map 裡的資料

假設你有一個像下面這樣的資料表。你想要找出其中出現次數最多的字，然後把這個字和次數印出來。

| 單字 | 次數 |
|---|---|
| Gonna | 3 |
| You | 3 |
| Give | 2 |
| Never | 1 |
| Up | 4 |

上面的單字出自 Rick Astley 的 1987 年金曲 『Never Gonna Give You Up』。

將這些資料整理成一個 map 集合：

```
words := map[string]int{
    "Gonna": 3,
    "You":   3,
    "Give":  2,
    "Never": 1,
    "Up":    4,
    }
```

用 for range 迴圈走訪它，記錄出現次數最多的單字，以及該單字的出現次數，然後印出以下結果：

**執行結果**

```
出現最多次的字: Up
出現次數      : 4
```

## 延伸習題 02.03：氣泡搜尋演算法

這次我們要對某個切片中的資料進行排序，做法是兩兩交換比對的元素，用暴力法一直執行到全部排好為只。這種排序技巧稱為氣泡排序法 (bubble sort)。

Go 語言其實有內建一個 sort 套件提供了排序演算法，但筆者希望讀者不要仰賴這個套件；此延伸習題的目標，是讓大家自己用剛學會的邏輯判斷和迴圈自行寫出演算法。

定義一個切片集合，其中有若干未經排序的數字。先把切片印到主控台上，將資料排序過後，再印出排序後的集合。

排序步驟如下：

1. 走訪切片，每次存取兩個相鄰的元素，若後面的比較小，就和前面的值對調，並記錄你做了對調動作。
2. 整個切片對調完一輪後，如果上一輪有做過對調，從頭重新再做一輪。
3. 等到整個切片已經由小到大排好，不須再做任何對調時，就結束排序。

在 Go 語言要對調兩個變數的值，可以這樣寫：

```
nums[i], nums[i-1] = nums[i-1], nums[i]
```

預期的輸出結果如下：

**執行結果**

```
排序前: [5 8 2 4 0 1 3 7 9 6]
排序後: [0 1 2 3 4 5 6 7 8 9]
```

## 延伸習題 3.01：銷售稅計算機

這個延伸習題要來寫一個專供購物車應用程式使用的工具，計算稅金並加總售價。

1. 建立一個計算單品稅金的程式。
2. 以售價和稅率作為計算機程式的輸入值。

3. 把稅金加總，再印出下表的稅金總額：

**購物清單及稅率**

| 項目 | 成本 | 稅率 |
|---|---|---|
| 蛋糕 | 0.99 | 7.5% |
| 牛奶 | 2.75 | 1.5% |
| 奶油 | 0.87 | 2% |

輸出應該像這樣：

**執行結果**

```
Sales Tax Total:  0.1329
```

## 延伸習題 03.02：房貸計算機

這次的活動要寫出一個專供線上財務顧問平台使用的貸款試算程式。計算規則如下：

1. 信用分數高於 450 者屬於信用良好，可享有 15% 的優惠年利率。
2. 信用分數低於 450 者，年利率為 20%。
3. 信用良好的人，每月付款金額最高不得超過每月收入的 20%。
4. 信用分數低於 450 者，每月付款金額最高不得超過每月收入的 10%。
5. 信用分數、每月收入、貸款金額、貸款期數小於 0 時必須示警（丟出錯誤）。
6. 貸款期數如果不能被 12 整除，也必須示警。
7. 利息計算方式為簡單的單利計算：貸款金額乘以年利率再除以 12, 即為每月支付利息。

計算完畢後，顯示貸款申請人詳情如下：

```
申請人 1
-----------
信用分數: 500
收入     : 1000
貸款金額: 1000
貸款利率: 15
貸款期數: 24
總利息   : 150
每月利息: 47.916666666666664
申請通過: true

申請人 2
-----------
信用分數: 350
收入     : 1000
貸款金額: 10000
貸款利率: 20
貸款期數: 12
總利息   : 2000
每月利息: 1000
申請通過: false
```

## 延伸習題 04:01：在陣列填滿值

在這個延伸習題，我們要定義一個陣列，然後用 for i 迴圈將其填滿。以下是步驟：

1. 定義一個有 10 個 int 元素的陣列。

2. 以 for i 迴圈將數字 10 到 19 填入陣列。

3. 用 fmt.Println() 將陣列內容顯示在主控台。

結果要像這樣：

**執行結果**

```
[10 11 12 13 14 15 16 17 18 19]
```

## 延伸習題 04.02：一週日子切片的輪轉

這個延伸習題要建立一個切片，並賦予一些資料值，然後我們要用先前學到的切片擷取知識來修改原本的切片：

1. 建立一段切片，把一週七天的名稱都放進去，從週一到週日：

```
"Monday", "Tuesday", "Wednesday", "Thursday", "Friday",          接下行
"Saturday", "Sunday"
```

2. 利用切片的範圍擷取寫法和 append() 更改切片，讓切片變成從週日開始、至週六結束。

3. 將切片顯示至主控台。

輸出應該像這樣：

**執行結果**

```
[Sunday Monday Tuesday Wednesday Thursday Friday Saturday]
```

## 延伸習題 04.03：從切片去掉一個元素

Go 語言沒有內建任何切片元素刪除功能，但你可以利用你所學到的技術做到類似的效果。這次就要建立一個含有資料的切片，而且要移除其中一個元素。做法不只一種，但各位讀者是否能以最精簡的方式完成它？

1. 建立一個切片，元素順序如下：

```
"Good", "Good", "Bad", "Good", "Good"
```

2. 利用切片的範圍擷取寫法和 append() 更改切片（提示：把切片切成兩半，重新接起來時漏掉 Bad 這個元素）。

3. 將新切片顯示至主控台。

## 延伸習題 04.04：查詢多重使用者

我們要定義一個 map, 並按照使用者輸入給程式的一系列鍵來顯示 map 中的相關資料：

1. 定義一個 map, 包含以下的鍵與對應值：

```
鍵：305, 值：Sue
鍵：204, 值：Bob
鍵：631, 值：Jake
鍵：073, 值：Tracy
```

2. 利用 os.Args 讀取傳給程式的多重鍵，然後把對應的人名印出來。如果某個鍵不存在於 map 中，也印出對應的訊息 ( 但不要直接結束程式 )。

3. 如果沒有任何鍵當成參數傳入，顯示錯誤並結束程式。

結果要像這樣：

**執行結果**

```
哈囉, Sue (305)
哈囉, Bob (204)
查無使用者 (632)
```

## 延伸習題 04.05：語系驗證器

我們要在這個延伸習題使用結構和 map 做一個語系驗證器，可以檢查你輸入的語系是否有支援。

語系由地區和語言兩部分組成，像是 "en_US"、"en_CN"、"fr_CN"、"fr_FR" 和 "ru_RU" 等。你需要做的事情如下：

❑ 建立 locale ( 語系 ) 結構型別，包含兩個欄位：language ( 語言，字串 ) 以及 territory ( 地區，字串 )。

❑ 寫一個函式來傳回一個 map, 其定義為 map[locale]bool。語系結構為鍵，至於值在此則不重要，我們只會檢查輸入的鍵是否存在於 map 內。

❑ 讀取使用者執行程式時輸入的語系參數。使用者會輸入 "en_US" 這樣的字串，你得將它分割成地區與語言兩部分：

```
localeParts := strings.Split(os.Args[1], "_")  // 以底線為界分割字串
    if len(localeParts) != 2 {  // 若分割後的切片長度不是 2, 代表語系格式錯誤
        fmt.Printf("語系輸入錯誤: %v\n", os.Args[1])
        os.Exit(1)
    }

    passedLocale := locale{  // 建立要查詢用的語系結構
        territory: localeParts[1],
        language:  localeParts[0],
    }
```

❖ 查詢 map, 檢查鍵是否存在，並回報使用者該語系是否有支援。

操作過程如下：

**執行結果**

```
PS F1741\Chapter04\Activity04.05> go run . "en_US"
支援傳入的語系
PS F1741\Chapter04\Activity04.05> go run . "en_FR"
不支援傳入的語系
```

## 延伸習題 04.06：型別檢查器

這個延伸習題的目標是檢查一個型別為 []interface{} 的切片，元素的型別各異。當中包括的型別如下：

❑ int

❑ float

❑ string

❑ bool

❑ struct

然後建立一個函式，可以接受任意一種型別。此函式會傳回一段字串，指出其接收的型別名稱：

❑ 如果是 int、int32、int64, 一律傳回 int。

❑ 如果是任何一種浮點數，一律傳回 float。

❑ 如果是字串，傳回 string。

❑ 如果是布林值，一律傳回 bool。

❑ 如果以上皆非，一律傳回 unknown。

預期的輸出如下：

**執行結果**

```
1 is int
3.14 is float
hello is string
true is bool
{} is unknown
```

## 延伸習題 05.01：計算程式設計師的工時

這個延伸習題要建立一個函式，可以計算公司內程式設計師的每週工時，以便拿來計算應付薪資。我們要用 struct 來記錄每位設計師的名字、時薪和一周每一天的工作時數：

```
type Developer struct {
    Name       string  // 名字
    HourlyRate int  // 時薪
    WorkWeek   [7]int  // 一周七天工時
}
```

為了方便查詢 WorkWeek 欄位，我們也定義一個自訂型別 Weekday (int), 以及對應到一周七天的常數：

```
type Weekday int

const (
    Sunday Weekday = iota // 0
    Monday                // 1 (以下類推)
    Tuesday
    Wednesday
    Thursday
    Friday
    Saturday
)
```

你得定義兩個函式，logHours() 以及 hoursWorked(), 使用第 4 章的值／指標接收器來讓它們變成 Developer 結構的方法：

1. func (d *Developer) logHours(day Weekday, hours int) 用來設定一星期某一天的工時。它接收一個 Developer 的指標型別 ( 傳入的結構則必須加上 &), 以便將特定日子的工時寫入該結構引數中。

2. func (d Developer) weekPayment() int 會計算傳入的 Developer 結構的一周總工時，並以此算出一周薪資。由於不須將資料寫回結構，因此這裡不需使用指標。

建立一個 Developer 型別的變數，指定程式設計師名字和時薪後，用 logHours() 登錄周一和周二若干工時 ( 並印出結構中欄位變更後的結果 ), 再用 weekPayment() 算出該周薪資：

**執行結果**

```
周一工時: 8
周二工時: 10
本周薪資: 180
```

## 延伸習題 05.02：計算程式設計師的工時——工時追蹤版

這個延伸習題承接前一個延伸習題。程式設計師一天可能會分幾段時間作業，我們要用閉包給他們提供工時追蹤功能，也就是個能累加工時的計數器，這樣他們打卡下班時就能用計數器來登錄當日工時。

你需要對前一個延伸習題做出的改變如下：

1. 在 Developer 結構新增一個欄位 WorkNonLogged, 其型別為閉包函式的型別。

2. 新增函式 nonLoggedHours(), 會傳回閉包函式，並包含一個工時計數器變數。閉包能接收一個整數引數，它會將計數器與該引數相加後傳回之，也就是累計但尚未正式登錄的工時。若若收到的引數為負值，則傳回目前的計數器值，並將計數器重設為 0。

3. logHours 不再需要傳入工時，直接用呼叫 WorkNonLogged 的閉包函式並傳入 -1 ( 重設追蹤工時 ) 即可。

4. 最後同樣顯示各天工時以及總薪資。

執行結果會像這樣：

**執行結果**

```
追蹤周一工時: 8
追蹤周二工時: 7
追蹤周二工時: 10
周一工時: 8
周二工時: 10
本周薪資: 180
```

## 延伸習題 6.01：為金融應用程式建立自訂錯誤訊息

有一間銀行希望在檢查姓氏和轉帳銀行／分行代碼 (routing number) 時，能加上一些自訂錯誤。他們發現，直接存入的程序會允許輸入無效的姓氏和轉帳代碼。銀行希望在發生這類問題時，可以印出清楚的訊息來說明錯誤。

1. 建立兩個 error 值，InvalidLastName 跟 InvalidRoutingNumbers。

2. 在 main() 函式中印出自訂訊息，好示範給銀行看。

預期輸出要像這樣：

**執行結果**

```
invalid last name
invalid routing number
```

## 延伸習題 06.02：驗證銀行客戶的輸入資料

銀行很滿意你在上一個練習中建立的自訂錯誤訊息，所以進一步要求你實作應用程式內容，用來驗證姓氏和轉帳代碼：

1. 建立一個名為 directDeposit 的結構。

2. directDeposit 含有三個字串型別欄位：lastName、firstName 和 bankName。此外還有兩個整數型別欄位：routingNumber 和 accountNumber。

3. directDeposit 結構要有一個 validateRoutingNumber() 方法，會在入帳金額少於 100 時傳回 ErrInvalidRoutingNumber。

4. directDeposit 結構還要有一個 validateLastName() 方法，會在姓氏為空字串時傳回 ErrInvalidLastName。

5. 在 main() 函式中將資料賦值給 directDeposit 結構的欄位，並逐一呼叫 directDeposit 結構的每一個方法。

預期的輸出要像這樣：

**執行結果**

```
invalid routing number
invalid last name
--------------------
姓:
名: Abe
銀行名稱: XYZ Inc
轉帳代碼:  17
帳號:  1809
```

## 延伸習題 06.03：輸入資料無效時引發 panic

延續習題 06.02，銀行現在決定，他們寧可在接收到無效轉帳代碼時讓程式當掉、好停止繼續處理直接存款。你必須拿上一個習題來改良：把 validateRoutingNumber 方法改成不要傳回 ErrInvalidRoutingNumber, 而是以該錯誤引發 panic。

預期輸出如下：

**執行結果**

```
panic: invalid routing number

goroutine 1 [running]:
main.(*directDeposit).validateRoutingNumber(...)
        main.go:45
main.main()
        main.go:30 +0x165
exit status 2
```

## 延伸習題 06.04：避免讓 panic 造成應用程式當掉

承上題，經過初步測試後，銀行不再堅持程式一定要當掉。因此這次我們要把上一個習題再加上一些功能，讓程式能從 panic 中復原、並印出造成 panic 的錯誤內容。

1. 在 validateRoutingNumber() 方法中新增一個用 defer 呼叫的匿名函式。
2. 在該匿名函式中用 if 檢查 recover() 是否傳回 error。如果有就印出來。

預期輸出如下：

**執行結果**

```
invalid routing number
invalid last name
--------------------
Last Name:
First Name:  Abe
Bank Name:  WilkesBooth Inc
Routing Number:  17
Account Number:  1809
```

## 延伸習題 07.01：計算年薪與工作考核分數

在這個習題，我們要來計算主管與程式設計師的年薪，以及後者的考核分數。主管與程式設計師的薪資計算規則不同，此外程式設計師的考核分數會來自多筆不同資料，用整數或文字的形式呈現。

1. 定義三個結構：Employee ( 員工 )、Developer ( 程式設計師 ) 及 Manager ( 主管 )，其中程式設計師與主管結構都會包含員工結構。另外也要定義一個介面 Payer。所有的定義如下：

```
type Employee struct {
    Id        int
    FirstName string
    LastName  string
}
```

```
type Developer struct {
    Individual        Employee
    HourlyRate        float64
    HoursWorkedInYear float64
    Review            map[string]interface{}
}

type Manager struct {
    Individual     Employee
    Salary         float64
    CommissionRate float64
}

type Payer interface {
    Pay() (string, float64)
}
```

2. 主管與程式設計師的資料建立如下：

```
// 程式設計師的考核資料
employeeReview := make(map[string]interface{})
employeeReview["WorkQuality"] = 5
employeeReview["TeamWork"] = 2
employeeReview["Communication"] = "Poor"
employeeReview["Problem-solving"] = 4
employeeReview["Dependability"] = "Unsatisfactory"

// 程式設計師
d := Developer{
    Individual:        Employee{Id: 1, FirstName: "Eric", LastName: "Davis"},
    HourlyRate:        35,
    HoursWorkedInYear: 2400,
    Review:            employeeReview,
}

// 主管
m := Manager{
    Individual:     Employee{Id: 2, FirstName: "Mr.", LastName: "Boss"},
    Salary:         150000,
    CommissionRate: .07,
}
```

3. 主管的薪資計算方式：m.Salary + (m.Salary * m.CommissionRate)

4. 程式設計師的薪資計算方式：d.HourlyRate * d.HoursWorkedInYear

5. 上面的 map『employeeReview』記錄了程式設計師的多筆考核資料，其值可能是整數或字串。我們要建一個字串考核的對照表如下：

```
var reviewStrToInt = map[string]int{
    "Excellent":      5,
    "Good":           4,
    "Fair":           3,
    "Poor":           2,
    "Unsatisfactory": 1,
}
```

6. 替 Employee 結構新增一個方法 FullName(), 會傳回員工的名 + 姓。

7. 替 Developer 和 Manager 各新增一個 Pay() 方法，並滿足 Payer 介面的規範。Pay() 方法會傳回程式設計師／主管的年薪。

8. 替 Developer 新增一個 ReviewRating() 方法，它會檢查其 Review 欄位 (map) 並算出平均考核分數。若考核資料是字串，就用上面的 reviewStrToInt 轉換。最後印出該員工的考核分數。

9. 撰寫一個 payDetails() 函式，接收數量不定的 Payer 介面型別，並呼叫每個值的 Pay() 方法，最後印出員工的全名與薪資。

預期的執行結果如下：

**執行結果**

```
Eric Davis 今年考績評為 2.80
Eric Davis 今年薪資為 84000.00
Mr. Boss 今年薪資為 160500.00
```

## 延伸習題 08.01：用套件計算年薪與工作考核分數

在這個課後練習中，我們要把延伸習題 07.01 改寫成模組化的版本，將 Employee、Developer 及 Manager 的型別及相關方法都移往各自的檔案中，並共同組成一個套件 payroll。

你該做的事情包括：

1. 在專案目錄下建立子資料夾 payroll, 內有 Employee.go、Devloper.go 和 Manager.go 三個檔案。會被外部套件 ( 即 main 套件 ) 用到的結構、結構欄位與方法須以大寫英文字母開頭來匯出，反之則應以小寫字母開頭來隱藏之。

2. 主程式 main.go (main 套件 ) 則置於專案目錄的 pay 資料夾下。

3. 用 go mod init < 專案名稱 > 替專案建立模組路徑。

4. main.go 會透過 < 模組路徑 >/payroll 來匯入 payroll 套件，並使用 init() 來初始化習題 07.01 中的 employeeReview、d 和 m 變數 ( 它們得改而宣告在套件層級 )。

預期輸出結果如下：

**執行結果**

```
初始化變數...
Eric Davis 今年考績評為 2.80
Eric Davis 今年薪資為 84000.00
Mr. Boss 今年薪資為 160500.00
```

## 延伸習題 09.01：建立社會安全碼驗證程式

在這個習題裡，我們要來驗證社會安全碼 (Social Security Numbers, SSN), 例如 123-45-6789。程式會先去掉破折號，然後要檢查 SSN 碼是否有效。就算資料無效，我們也不希望中斷程式，而是輸出日誌來回報錯誤，

並繼續檢查下一個 SSN 碼。

你可使用 string.Replace() 來將字串內的破折號換成空字元：

```
strings.Replace(ssn, "-", "", -1)
```

SSN 碼去掉破折號後的有效規則如下：

- ❑ 長度必須為 9 位數
- ❑ 內容必須都是數字
- ❑ 開頭不能是 000
- ❑ 若開頭為 9, 第 4 位數必須為 7 或 9

你要做的事情包括建立自訂 error 值，以及四個檢查 SSN 碼格式的函式。這些函式在檢查失敗時會用 fmt.Errorf() 傳回包含有對應 error 值的錯誤。而 main() 若收到錯誤，就用 log 套件將之輸出到主控台。

需檢查的範例資料為字串切片：

```
validateSSN := []string{"123-45-6789", "012-8-678", "000-12-0962",
"999-33- 3333", "087-65-4321","123-45-zzzz"}
```

其輸出結果應類似如下：

**執行結果**

```
2021/04/19 15:10:35.175342 main.go:23: 驗證資料 6 之 1: "123-45-6789"
2021/04/19 15:10:35.229966 main.go:23: 驗證資料 6 之 2: "012-8-678"
2021/04/19 15:10:35.230496 main.go:31: 值 0128678 導致錯誤: SSN 不足 9 字元
2021/04/19 15:10:35.230496 main.go:23: 驗證資料 6 之 3: "000-12-0962"
2021/04/19 15:10:35.231086 main.go:35: 值 000120962 導致錯誤: SSN 以 000 開頭
2021/04/19 15:10:35.231086 main.go:23: 驗證資料 6 之 4: "999-33-3333"
2021/04/19 15:10:35.231600 main.go:39: 值 999333333 導致錯誤: SSN 以 9 開頭時
                                       第 4 位需為 7 或 9
2021/04/19 15:10:35.232133 main.go:23: 驗證資料 6 之 5: "087-65-4321"
```

```
2021/04/19 15:10:35.232198 main.go:23: 驗證資料 6 之 6: "123-45-zzzz"
2021/04/19 15:10:35.232724 main.go:27: 值 12345zzzz 導致錯誤: SSN 非數字
```



## 延伸習題 10.01：計算保險申請期限

因應客戶需求，現在你要撰寫一支程式，能夠根據指定的時間和時區計算保險給付申請的期限 (5 年) 將在何時到期。你得撰寫一個函式，它會接收一個時間值及一個 IANA 時區識別名稱，以此計算出該時區的到期時間。

```
deadline(t time.Time, tz string) (time.Time, error)
```

這個函式會將 t 加上 5 年，轉換成指定的時區然後傳回。若時區名稱不正確，就傳回原始的時間值以及 error 值。

另外，客戶也要求以『時 : 分 : 秒 月 / 日 / 年』的格式顯示時間，因此你也得將時間做格式化。

若以系統時間和東京時區 (Asia/Tokyo, 比台北快 1 小時) 為輸入資料，此程式的預期輸出結果會類似如下：

```
現在時間: 16:18:15 04/23/2021
期限     : 17:18:15 04/23/2026
```

## 延伸習題 10.02：測量程式執行時間

在這個習題，我們要對一個演算法函式撰寫一個單元測試，它會測試該函式一千次，並測量每次執行的時間。如果函式傳回錯誤結果，就立即中斷測試。

你要測試的演算法能計算費伯納奇數列 (Fibonacci sequence) 中的某個數字：

```
// Fibonacci calculates a value in the Fibonacci sequence
func Fibonacci(n int) int {
    if n < 2 {
        return n
    }
    return Fibonacci(n-1) + Fibonacci(n-2)  // 遞迴呼叫
}
```

目前已知 Fibonacci(30) 會得到 832040。將上面這個函式放在 main.go 中，接著撰寫 main_test.go, 在裡面建立單元測試函式：

```
func TestMain(t *testing.T) {
    // 測試程式碼
}
```

這個函式內要測試 Fibonacci() 一千次，並用 log.Println() 印出每次執行 Fibonacci(30) 所花的時間 ( 比如用 Duration.Seconds())。假如 Fibonacci(30) 的傳回值不是 832040, 就用 t.Fatal() 中止測試。

執行結果應該要類似如下 ( 執行時間取決於您的系統 )：

**執行結果**

```
PS F1741\Chapter10\Activity10.02> go test -v .
=== RUN   TestMain
... (中略)
2021/04/23 18:01:03 0.0104838
2021/04/23 18:01:03 0.0105248
2021/04/23 18:01:04 0.0091036
2021/04/23 18:01:04 0.0129523
2021/04/23 18:01:04 0.0116262
2021/04/23 18:01:04 0.0099896
2021/04/23 18:01:04 0.009972
2021/04/23 18:01:04 0.0113049
2021/04/23 18:01:04 0.0091475
2021/04/23 18:01:04 0.0118717
--- PASS: TestMain (11.05s)
PASS
ok      F1741/Chapter10/Activity10.02   12.113s
```

# 延伸習題 11.01：產生客戶與訂單資料的 JSON 格式

我們要來模擬一個線上購物網站，接收到使用者的基本資訊 JSON 資料，並把使用者放入購物車的商品加到該 JSON 資料中，以便傳給結帳程式處理。

一開始傳入的 JSON 字串會像下面這樣：

```
{
    "username":"blackhat",
    "shipto":{
        "street":"Sulphur Springs Rd",
        "city":"Park City",
        "state":"VA",
        "zipcode":12345
    },
    "order":{
        "paid":false,
        "orderdetail":[]
    }
}
```

程式解析 JSON 後，會在鍵 orderdetail 底下新增三件商品的訂單細節。重新產生的 JSON 字串會變成如下：

```
{
    "username": "blackhat",
    "shipto": {
        "street": "Sulphur Springs Rd",
        "city": "Park City",
        "state": "VA",
        "zipcode": 12345
    },
    "order": {
        "total": 475,
        "paid": true,
        "fragile": true,
        "orderdetail": [
            {
```

```
                "itemname": "A Guide to the World of zeros and ones",
                "desc": "book",
                "qty": 3,
                "price": 50
            },
            {
                "itemname": "Final Fantasy The Zodiac Age",
                "desc": "Nintendo Switch Game",
                "qty": 1,
                "price": 50
            },
            {
                "itemname": "Crystal Drinking Glass",
                "qty": 11,
                "price": 25
            }
        ]
    }
}
```

你要做的事情如下：

1. 替以上的 JSON 資料建立對應的 Go 語言結構，以便把 JSON 解碼成我們能處理的形式。

2. 替最上層的結構 (customer) 新增兩個方法 AddItem() 和 Total(), 前者用來新增一個商品訂單，後者用來統計所有商品的總額 ( 鍵 total), 並標明是否已付款 ( 鍵 paid)、是否為易碎貨品 (fragile)。

3. 將結構轉成 JSON 字串並印出。

## 延伸習題 12.01：解析銀行交易 CSV 檔

在這個練習中，我們要解析一個銀行交易檔案，其內容為 CSV 格式，當中的欄位包括交易代碼 (id)、收款人 (payee)、支出金額 (spent) 和支出預算分類 (budget category)。檔案內容如下：

**bank.csv**

```
id, payee, spent, category
1, sheetz, 32.45, fuel
2, martins, 225.52, gaming
3, wells fargo, 1100, mortgage
4, joe the plumber, 275, repairs
5, comcast, 110, tv
6, bp, 40, fuel
7, aldi, 120, food
8, nationwide, 150, car insurance
9, nationwide, 100, life insurance
10, jim electric, 140, utilities
11, propane, 200, utilities
12, county water, 100, utilities
13, county sewer, 105, utilities
14, 401k, 500, retirement
```

其中 CSV 檔的預算分類必須依據右表來轉換：

| 來源 | 目標 |
|---|---|
| fuel, gas | autoFuel |
| mortgage | mortgage |
| repairs | repairs |
| car insurance, life insurance | insurance |
| utilities | utilities |

假如 CSV 檔中列出的預算無法轉換，則回報『查無分類』錯誤。

本練習的目的在於建立一隻命令列程式，可以接收兩個旗標：-source 為銀行交易 CSV 檔的路徑與檔名、-log 則是日誌檔，後者負責記錄 CSV 解析過程中發生的錯誤。如果使用者沒有輸入這兩個旗標，程式會直接停止並提示旗標的用法。若 CSV 檔不存在，程式同樣會直接結束。

若兩個旗標值都有正確提供，一個函式會解讀 CSV 檔的交易記錄、將之整理成結構變數切片並傳回，再由 main() 走訪並格式化印出。在解析過程中，任何錯誤除了用 log 顯示在主控台之外，也會建立一個 logger 將錯誤及相關訊息寫入使用者提供的日誌檔 ( 假如不存在就建立一個 )。日誌訊息會以附加方式寫入日誌檔，以免覆蓋掉之前存在的記錄。

以下是此練習預期的輸出結果：

**執行結果**

```
PS F1741\Chapter12\Activity12.01> go run . -source bank.csv -log
bank.log
2021/05/04 10:34:46.183658 F1741/Chapter12/Activity12.01/main.
go:149: 查無分類 [Id: 2  gaming]
2021/05/04 10:34:46.185799 F1741/Chapter12/Activity12.01/main.
go:149: 查無分類 [Id: 5  tv]
2021/05/04 10:34:46.187060 F1741/Chapter12/Activity12.01/main.
go:149: 查無分類 [Id: 7  food]
2021/05/04 10:34:46.188273 F1741/Chapter12/Activity12.01/main.
go:149: 查無分類 [Id: 14  retirement]
CSV 檔解析結果:
Id: 1   Payee: sheetz           Spent: 32.45    Cateory: autoFuel
Id: 2   Payee: martins          Spent: 225.52   Cateory:
Id: 3   Payee: wells fargo      Spent: 1100     Cateory: mortgage
Id: 4   Payee: joe the plumber  Spent: 275      Cateory: repairs
Id: 5   Payee: comcast          Spent: 110      Cateory:
Id: 6   Payee: bp               Spent: 40       Cateory: autoFuel
Id: 7   Payee: aldi             Spent: 120      Cateory:
Id: 8   Payee: nationwide       Spent: 150      Cateory: insurance
Id: 9   Payee: nationwide       Spent: 100      Cateory: insurance
Id: 10  Payee: jim electric     Spent: 140      Cateory: utilities
Id: 11  Payee: propane          Spent: 200      Cateory: utilities
Id: 12  Payee: county water     Spent: 100      Cateory: utilities
Id: 13  Payee: county sewer     Spent: 105      Cateory: utilities
Id: 14  Payee: 401k             Spent: 500      Cateory:
```

以上輸出的前四行內容 ( 錯誤訊息 ) 也會寫入到 bank.log 中。

## 延伸習題 13.01：解析銀行交易 CSV 檔並寫入資料庫

以延伸習題 12.01 為基礎，這回程式在解析 CSV 檔之後，會將它寫入資料庫：

1. 在一開始時提示使用者輸入資料庫帳號及密碼，若任一沒輸入就直接結束程式。解答會沿用書中的 MySQL 資料庫設定及驅動程式。

2. 新建一個資料表，其名稱即為輸入的 CSV 檔名 ( 不含 .csv)。為了確保能正確寫入，也得在新建資料表之前先嘗試刪除它。

3. 將程式從 CSV 檔解析出來的資料，一列列寫入該資料表。注意：若有任一欄位為零值 ( 解析時有問題 ), 就不寫入資料庫。

4. 讀取該資料表，並以和習題 12.01 相同的方式格式化印出。

5. 替資料庫的新建資料表、插入資料和查詢資料建立相關的自訂 error, 並沿用習題 12.01 的日誌功能。

你能用以下程式碼來讓使用者從主控台輸入帳號及密碼。注意輸入密碼的部分我們使用了 ssh/terminal 套件 (golang.org/x/crypto/ssh/terminal, 請用 go get 下載 ), 這能讓使用者輸入的密碼不會顯示在主控台中。

```
var userName string
fmt.Print("請輸入資料庫帳號: ")
fmt.Scanln(&userName)
if len(userName) == 0 {
    fmt.Println("未輸入資料庫帳號")
    os.Exit(1)
}

fmt.Print("請輸入資料庫密碼: ")
password, err := terminal.ReadPassword(int(syscall.Stdin))
if err != nil {
    fmt.Println("未輸入資料庫密碼")
    os.Exit(1)
}
fmt.Println("")
```

執行結果會類似如下，注意到解析過程有錯的資料不會被寫入資料表中：

```
請輸入資料庫帳號: user
請輸入資料庫密碼:
2021/05/15 15:12:00.599905 F1741/Chapter13/Activity13.01/main.go:235:
查無分類 [Id: 2  gaming]
2021/05/15 15:12:00.600421 F1741/Chapter13/Activity13.01/main.go:235: 查
```

```
無分類 [Id: 5  tv]
2021/05/15 15:12:00.600938 F1741/Chapter13/Activity13.01/main.go:235: 查
無分類 [Id: 7  food]
2021/05/15 15:12:00.601456 F1741/Chapter13/Activity13.01/main.go:235: 查
無分類 [Id: 14  retirement]
DB 查詢結果:
Id: 1   Payee: sheetz           Spent: 32.45    Cateory: autoFuel
Id: 3   Payee: wells fargo      Spent: 1100     Cateory: mortgage
Id: 4   Payee: joe the plumber  Spent: 275      Cateory: repairs
Id: 6   Payee: bp               Spent: 40       Cateory: autoFuel
Id: 8   Payee: nationwide       Spent: 150      Cateory: insurance
Id: 9   Payee: nationwide       Spent: 100      Cateory: insurance
Id: 10  Payee: jim electric     Spent: 140      Cateory: utilities
Id: 11  Payee: propane          Spent: 200      Cateory: utilities
Id: 12  Payee: county water     Spent: 100      Cateory: utilities
Id: 13  Payee: county sewer     Spent: 105      Cateory: utilities
```

## 延伸習題 14.01：從網路伺服器請求資料並處理回應

想像你在跟一個網路 API 打交道，送出一個 GET 請求來取得 JSON 資料，並在解讀後得到一個由重複人名組成的陣列。你得計算每個名字出現幾次。

1. 建立一個資料夾，例如 Activity 14.01，並在其下建立兩個子目錄 server 與 client。

2. 在 server 子資料夾內建立檔案 server.go 並撰寫伺服器程式．它會傳回隨機次數的 Electric 和 Boogaloo 這兩個名字：

```
{
    "name": ["Electric", "Electric", "Electric", "Boogaloo", "Boogaloo", ...]
}
```

3. 在 client 子資料夾內建立檔案 main.go 並撰寫客戶端程式，當中的 getDataAndParseResponse() 函式會解析 JSON 並傳回兩個整數，代表 Electric 和 Boogaloo 出現的次數。

4. 執行伺服器與客戶端程式，讓客戶端印出結果（名字的出現次數每次都會不同）：

```
Electric Count:  4
Boogaloo Count:  1
```

## 延伸習題 14.02：驗證使用者登入帳密及授權

在這個習題中，你要練習實作一個能驗證使用者提供的授權碼、登入帳號和密碼是否正確的伺服器程式，並用 log 套件記錄／傳回使用者試圖登入的結果：

1. 客戶端程式會送出 POST 請求，請求標頭的 "Authorization" 欄位會填入授權碼，而請求主體則夾帶使用者帳密。使用者帳密為 JSON 格式資料，當中包含兩個欄位：ID（帳號）和 password（密碼）。

2. 伺服器收到請求後，首先判斷標頭的 "Authorization" 欄位是否為正確的授權碼，接著解析主體內的 JSON 資料，看看使用者的帳號或密碼是否存在。以上三者都通過時才算登入成功。為了模擬起見，授權碼以常數寫死在伺服器中，使用者帳密則以 map[string]string 變數形式寫死在伺服器內。

3. 不論登入成功與否，伺服器會以 log 套件印出相關訊息、以自訂 logger 將訊息記錄在日誌檔中，並將同樣的訊息傳回給客戶端。

以下是客戶端程式的部分內容：

```
func main() {
    user := Account{ID: "John", Password: "1234"}
    token := "AuthorizedUser1234"
    result := loginServer(user, token)
    log.Printf("Login status: %v\n", result.Status)
}
```

以上程式執行後，伺服器端應顯示類似以下訊息，而同樣的訊息會記錄在日誌檔 ( 比如 login.log) 中：

```
2021/05/10 13:58:00.924159 F1741/Chapter14/Activity14.02/server/server.
go:77: login successful: John
```

## 延伸習題 15.01：帶有計數器的電子書網站

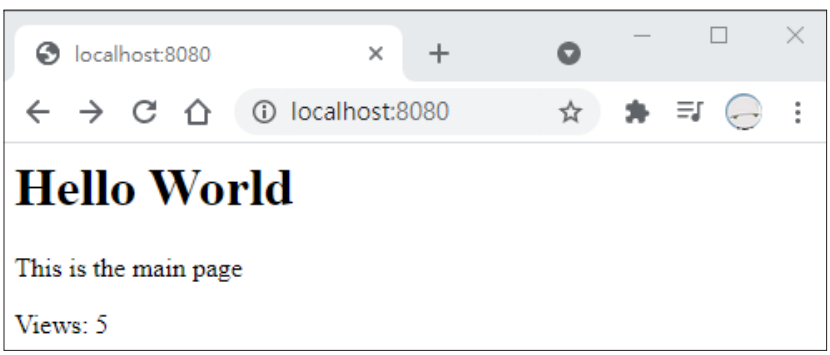
建立一個能顯示電子書內容的網站，它有以下三個路徑可以請求：

| | |
|---|---|
| 首頁 | http://localhost:8080/ |
| 第一章 | http://localhost:8080/chapter1 |
| 第二章 | http://localhost:8080/chapter2 |

每個頁面的內容都用一個 PageWithCounter 結構代表，記錄該頁的瀏覽計數器、標題及內容：

```
type PageWithCounter struct {
    counter int
    heading string
    content string
}
```

這三個結構所使用的 ServeHTTP() 方法，會根據結構內容傳回不同的網頁，並每次將該結構的計數器加 1。下圖以首頁為例：



注意

注意這支程式並未考慮多人同時操作的情境；第 16 章會再談到如何用互斥鎖確保資料安全。

各位寫出程式後，可能也會發現計數器每次會增加 2 而不是 1。這是因為瀏覽器也會試圖存取網站圖示，以致瀏覽次數多了一次。解決方式是註冊以下路徑 /favicon.ico，讓它不做任何事情：

```
http.HandleFunc("/favicon.ico", func(w http.ResponseWriter,  接下行
r *http.Request) {})
```

## 延伸習題 15.02：對 HTTP 請求回應 JSON 資料

修改延伸習題 15.01，使伺服器用 json.Marshal() 把結構轉換成 JSON，而不是傳回 HTML 格式內容：

```
{"views":5,
 "title":"Hello World",
 "content":"This is the main page"}
```

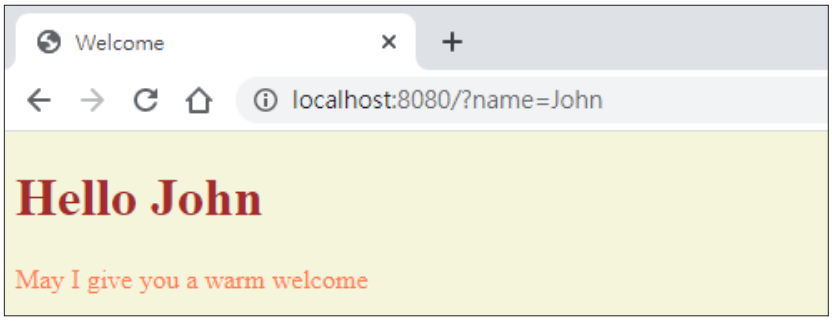
若轉換錯誤，傳送 HTTP 代碼 400 (bad request) 給使用者：

```
w.WriteHeader(400)
```

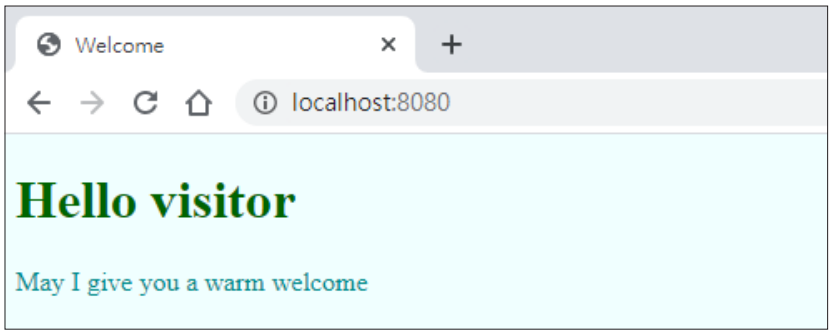
## 延伸習題 15.03：使用外部模板動態載入不同的 CSS 檔

在練習 15.06 中，我們看到如何透過檔案伺服器來分享 CSS 檔案。這回你要寫一個會顯示歡迎訊息的網站，使用者可決定要不要在網址後面加上 name 參數。伺服器會根據 name 參數存在與否顯示不同的歡迎訊息，甚至載入不同的 CSS 檔案。

1. 如果網址有 name 參數，會顯示以下結果 ( 載入 named.css)：




2. 如果未提供 name 參數，則顯示以下結果 ( 載入 visitor.css)：




CSS 檔需放在專案的 ./public 子目錄下，而在伺服器的存取路徑則是在 /statics/ 路徑下。named.css 與 visitor.css 的內容相同，只是色彩不一樣而已：

named.css

```
body {
    background-color: beige;
}

h1 {
    color: brown;
}

p {
    color: coral;
}
```

visitor.css

```
body {
    background-color: azure;
}

h1 {
    color: darkgreen;
}

p {
    color: darkcyan;
}
```

## 延伸習題 16.01：使用互斥鎖的累加

請改寫書中的 Exercise 16.03, 同樣新增 4 個 Goroutine, 處理的數字範圍分別為 1~25、26~50、51~75、76~100：

1. 使用 sync.Mutex 互斥鎖 ( 見 16-3-2 節 )
2. sum() 函式需接收 *sync.Mutex 型別的指標
3. 現在 sum() 的 res 指標 ( 以及從 main() 傳入的變數 ) 要改成 string 型別 , 它會累加一個新字串上去：

```
|數字|
```

- 你可以用以下方式來產生格式化字串：

```
fmt.Sprintf("|%d|", i)
```

執行結果應該如下 ( 你看到的數字順序應會略有差異 )：

**執行結果**

```
2021/03/10 15:15:50 |76||1||2||3||4||5||6||7||8||9||10||11||12||13||14||1
5||16||17||18||19||20||21||22||23||24||25||77||78||79||80||81||82||83||84
||85||86||87||88||89||90||91||92||93||94||95||96||97||98||99||100||26||27
||28||29||30||31||32||33||34||35||36||37||38||39||40||41||42||43||44||45|
|46||47||48||49||50||51||52||53||54||55||56||57||58||59||60||61||62||63||
64||65||66||67||68||69||70||71||72||73||74||75|
```

## 延伸習題 16.02：以 Goroutine 處理來源檔案

你在這個延伸習題中要寫一支程式，同時讀取兩個含有一些數字的文字檔，用管線 (pipeline) 模式傳給其他程序處理，統計出偶數和奇數數字的總和，並將這結果寫入另一個檔案中。

例如，若文字檔內容分別為

**input1.dat**

```
1
2
5
```

和

**input2.dat**

```
3
4
6
```

輸出結果應該為

**result.txt**

```
Even 12
Odd 9
```

程式中會包含以下的 Goroutine：

- source() 用來讀取檔案，並把數字（每一行去掉換行符號後轉成整數）寫入通道 data。
- splitter() 從通道 data 讀取數字，並將偶數存入通道 even, 將奇數存入通道 odd。
- sum() 可從 even 或 odd 讀取數字並加總，存入通道 sumeven 或 sumodd。
- merger() 將 sumeven 和 sumodd 的值寫入結果檔案。
- 兩個 source() 共用一個 WorkGroup，其他程序則共用另一個 WorkGroup。在 main() 結尾先等待 source() 都結束，再等待其他程序結束。