

Go Programming



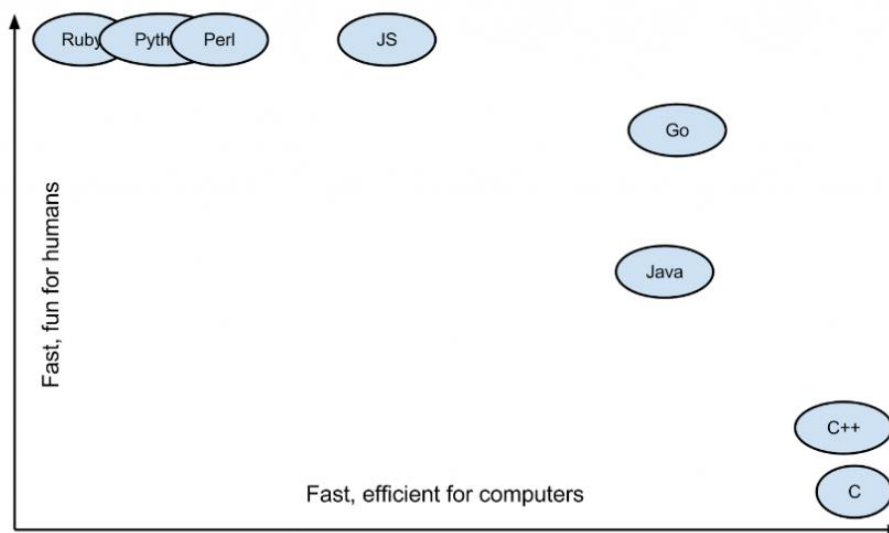
Go 語言

- Go語言的原本名稱為Go Language，因此又稱Golang，之後文章簡稱Go，是Google在2009年推出的一個實驗性項目的程式語言，會出現這個語言的原因之一是因為Google工程師們不是那麼喜歡使用C++。Go語言在2007年開始設計、2009年正式公開、2011年釋出v1.0版本，之後大約每半年的時間釋出下一個版本，如今已來到v1.19 (2022/08)。
- Golang 一開始是由C語言編寫而成。**執行程式需要經過編譯，為強型別程式語言**。跟C語言有點類似、也具有**指標**功能，如果有學過C語言的很容易上手。也有人說Go是現代版的C語言，冠上如此華麗的美稱，風格與C語言相似，有"網路的C語言"的稱號。但無論如何，想用來取代C語言終究是有一定難度的。
- 效能比Python好，但略比C差，然後常常被跟Rust比來比去。內建**併發**，很容易就能寫出**多線程**程式。代表能榨乾CPU的資源，或者說是以最少的資源做最多的事、不浪費每顆CPU。
- 跨平台、容易部署，**內建垃圾回收機制**。而且簡單易學。

Go語言

- 為甚麼選擇Golang？
 - 在執行效率上，C語言表現很好，但是很難讓人看懂，在易讀性中，Ruby, Python位居好讀榜首，但執行效率總是不太好。那有沒有好讀又跑得快的程式語言呢？可能會回答說：魚與熊掌不可兼得，但是Google辦到了！那就是Go：

After Go



source: [Brad Fitzpatrick](#)

- 從圖中可以看到，Golang可以說是囊括了兩種優勢。

Go 語言

- Go的風格很像C語言，尤其是一樣用*表示指標，而Go也做了一些語法上的簡化及統一，使得程式在開發時更容易上手，以及更容易閱讀，以下是一些主要的差異：
 1. 每一行程式的結尾不需要分號 (;)
 2. 大括號 { } 的左括號 { 不能換行寫
 3. for, if, switch條件式不需要小括號 ()
 4. 使用 tab 來排版。
- Go的特性與優點應該是已經被講到爛掉的東西，不外乎就是編譯快、效能高、容易部屬、語法簡單及內建併發之類的，以下列出的是常提到的特性及優點：
 1. 環境簡單，跨平台，上手容易。
 2. 編譯輸出可執行文件，部屬快速容易。
 3. 編譯(靜態編譯)速度快，執行效能高，語法卻如同腳本語言輕快。
 4. 語言層原生支援併發(goroutine)。
 5. 開發工具內建性能分析。
 6. 強大的標準函式庫(stdlib)。
 7. 程式碼風格清晰、簡單(保留字只有25個)，並且有強制性。
 8. gofmt，官方指定 coding style，使得可讀性更佳。

Go 語言

- 什麼是靜態程式語言？

- 簡單來說，靜態語言的意思，就是當宣告一個變數時，就需要同時宣告此變數會存放的資料型態為何；反之，動態語言則不需要在宣告變數時，就確定存放的資料型態。
- 大家比較熟悉的靜態語言有：C、Java、**Golang** 等，動態則是：JavaScript、**Python**、Ruby
- 用一些程式碼來舉例：

```
// 變數 age 存放 value 3  
var age int = 3  
age = "jimmy"  
// cannot use "jimmy" (untyped string constant) as int value in assignment
```

```
## 將 345 放進 age 變數  
age = 345  
## 將 age 變數改成放 string  
age = "jimmy"
```

- 從以上兩個例子可以看出來，在 Go(左圖)裡面，當變數被賦值時，此變數能存放的資料型態也被定義了，當改變時，就會報錯，反之 Python(右圖) 則是可以任意更動。
- 靜態與動態語言各有好處，最主要的差別在於靜態語言是事先定義好資料型態，所以程式在執行時，不需要多花效能來判斷資料型態，可以提升程式的執行速度。而動態語言的好處，則是撰寫簡單，不需要在宣告變數時，就寫定其資料型態，對於新手來說相對簡單好用。

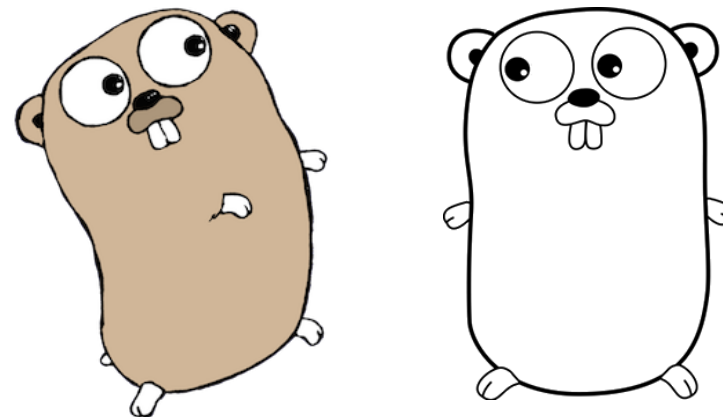
Go 語言

- 什麼是編譯式語言？
 - 編譯式與直譯式語言最大的差別，在於程式執行方式不同。
 - 編譯式：當寫完程式腳本時，需要先將腳本 **compile(編譯)** 成電腦懂的腳本，在將整包腳本拿去執行。
 - 直譯式：當寫完程式腳本時，直接使用直譯器一行一行翻譯成電腦語言並執行。
 - 主要的優缺點就執行特性相關。因為編譯式是整包打包轉譯、運算，而直譯是一行一行翻譯再執行所以：
 1. 編譯式執行效率較佳
 2. 直譯式相對容易 Debug
 3. 編譯式語言編譯完的腳本，可以直接在各類 OS 系統中執行，因為其編譯完的腳本，就是電腦懂的語句。
 4. 直譯式則需要在特定的執行載體才能在電腦內執行相對應的腳本

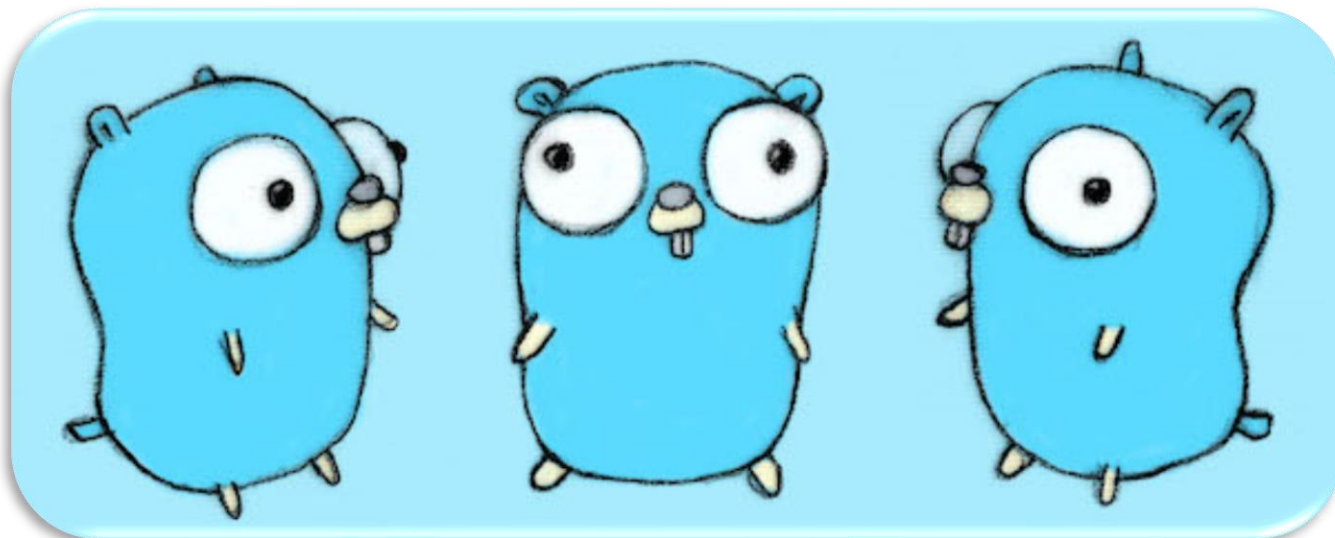
Go語言

- Go語言有著一堆很要求的規定，稍微提一下幾個特點：
 1. 變數宣告了就要用！不然編譯器爆錯！
 2. 套件import就要用！
 3. 大括號排版都要照格式
 4. 還有gofmt(go format)可以幫助排版，固定格式
 5. 變數字開頭大寫代表開放外界使用
 6. 充斥大量的 `if err != nil{ }`，判斷錯誤的方法。發生預期之外的事情就返回、跳出往下執行的程式是預期中的事情
 7. ...
- Google對於程式風格的講究，推了一大堆嚴謹的命名規範與慣例，反正Google是軟體界大佬，好的Coding習慣，多少就遵守。
- 缺點：
 1. 垃圾回收機制一直被人詬病。
 2. 語言中不使用泛型與異常處理。

Go 語言



- Go 可以做什麼：
 - 簡而言之，Go 語言穩定，也很容易上手，工程師們寫出的程式碼風格較一致，常被拿來架設伺服器以及網站後端。
 - 除此之外，聽過 **Docker**、**Kubernetes** 嗎？這兩個鼎鼎大名的 **容器** 專案是使用 Go 語言編寫的產物之一。也可以看看在 Github 開源社區中的星星排名與 Go 百大專案。
 - 社群人數正往上攀升成長、成為最具前途及錢途的程式語言之一。



Go語言

- 有哪些企業用了Golang：

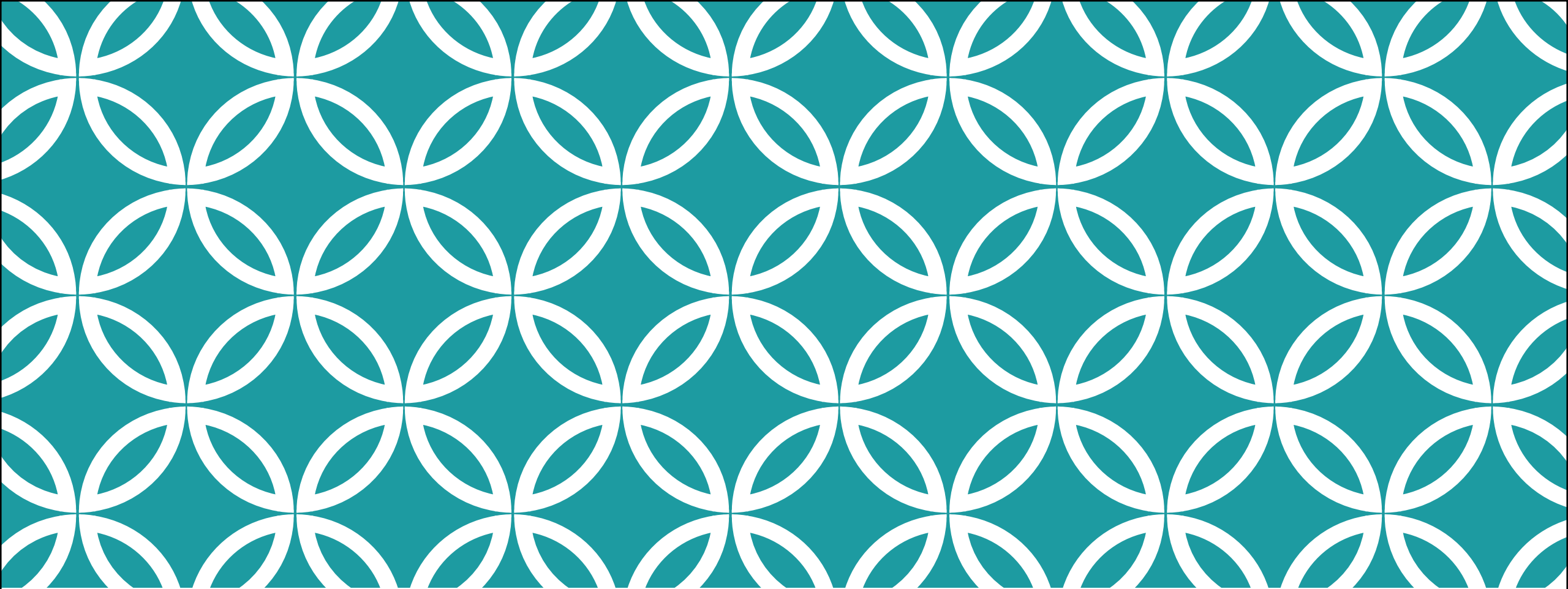


- 其他更多使用Golang的企業：<https://github.com/golang/go/wiki/GoUsers>
- 企業怎麼用Golang：<http://techstacks.io/tech/go>

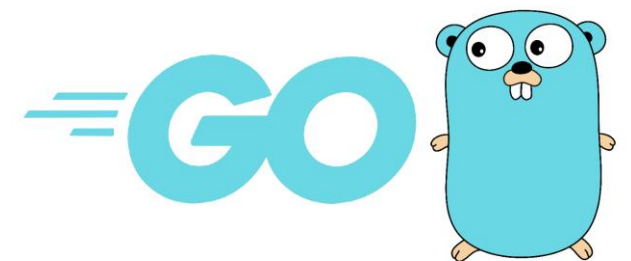
Go 語言

Golang 相關學習資源

- [The Go Programming Language](#)
 - Golang 官方網站，可以線上執行、分享程式碼、參考文件等相關內容
- [A Tour of Go](#)
 - 官網下的功能，可以說是Golang練功房
- [Go by Example](#)
 - 各種Golang程式碼範例
- [An Introduction to Programming in Go | Go Resources](#)
 - 另一個Go教學
- [GoDoc](#)
 - Go packages的文件，包含opensource的package



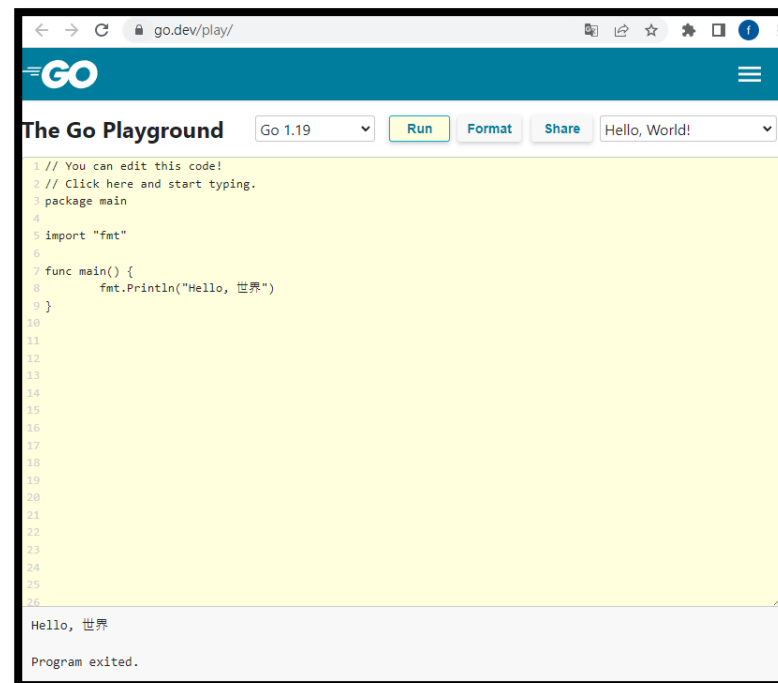
Start to Go



Start to Go

Go Playground

- 連結點開來就有一隻現成的Hello World可以執行了。
- 有這個Go Playground十分方便，可以在這平台測試跑小程式、也可以Share程式碼、複製網址貼給其他人看。
- 注意：
 - 在這個平台沒辦法透過Scanf讀取鍵盤輸入
 - 時間也是被固定的，怎麼random跑出來的亂數也是同一組



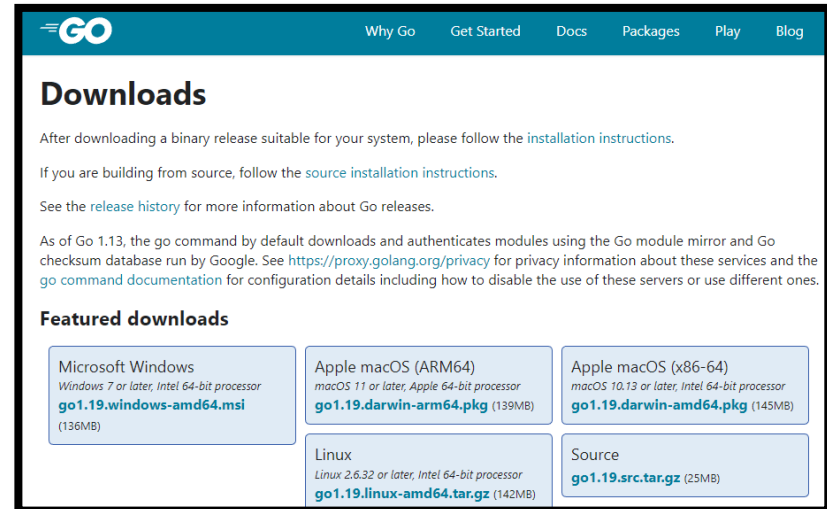
Start to Go

安裝Go語言

- 非常的簡單，只要在[這裡](#)下載，看到上圖，go支援三種作業系統Windows, Mac, Linux，其中Windows/MacOS只要下載檔案並且執行就能完成安裝，下一步下一步安裝就可以了。
- 預設的 Go的根目錄 (GOROOT) 會在：
`C:\go` (for Windows)
`/root/go` (for Linux/MacOS)
底下，裡頭包含執行go語言基本必備的官方library。預設的 Go專案路徑 (GOPATH) 則是在：
`C:\Users\USER\go` (for Windows)
`/Users/USER/go` (for Mac, Linux)
- 在Go後來的版本中(1.10之後)不用手動設置GOPATH和GOROOT，安裝完畢後，可以透過指令來確認是否安裝完成(注意大小寫)：

```
> go
> go version

> echo %GOPATH%
> go env GOPATH
> go env GOROOT
```



Start to Go

安裝Go語言

Windows版本：

```
(base) C:\Users\Alvin>go
Go is a tool for managing Go source code.

Usage:

    go <command> [arguments]

The commands are:

    bug          start a bug report
    build        compile packages and dependencies
    clean        remove object files and cached files
    doc          show documentation for package or symbol
    env          print Go environment information
    fix          update packages to use new APIs
    fmt          gofmt (reformat) package sources
    generate     generate Go files by processing source
    get          add dependencies to current module and install them
    install     compile and install packages and dependencies
    list        list packages or modules
    mod         module maintenance
    work        workspace maintenance
    run         compile and run Go program
    test        test packages
    tool        run specified go tool
    version     print Go version
    vet         report likely mistakes in packages

Use "go help <command>" for more information about a command.

Additional help topics:

    buildconstraint build constraints
    buildmode       build modes
    c               calling between Go and C
    cache           build and test caching
    environment     environment variables
    filetype        file types
    go.mod          the go.mod file
    gopath          GOPATH environment variable
    gopath-get      legacy GOPATH go get
    goproxy         module proxy protocol
    importpath      import path syntax
    modules         modules, module versions, and more
    module-get      module-aware go get
    module-auth     module authentication using go.sum
    packages        package lists and patterns
    private         configuration for downloading non-public code
    testflag        testing flags
    testfunc        testing functions
    vcs             controlling version control with GOVCS

Use "go help <topic>" for more information about that topic.

(base) C:\Users\Alvin>go version
go version go1.19 windows/amd64

(base) C:\Users\Alvin>echo %GOPATH%
C:\Users\Alvin\go

(base) C:\Users\Alvin>go env GOPATH
C:\Users\Alvin\go

(base) C:\Users\Alvin>go env GOROOT
C:\Program Files\Go
```

MacOS版本：

```
Apple ~ go version
go version go1.14.2 darwin/amd64
Apple ~ echo $GOPATH
/Users/g2020070116/go
Apple ~ go env GOPATH
/Users/g2020070116/go
Apple ~ go env GOROOT
/usr/local/go
Apple ~
```

- 可以看到有go的用法跑了出來，而不是 command not found 等字樣，
- 如果GOPATH或GOROOT的路徑跑不出來，再手動添加環境變數即可。

Start to Go

安裝Go語言

- 第一個Go程式：
 - 通常學習新語言的第一支程式都是同一個情境，就是啟動程式，然後印出hello world。首先在GOPATH底下建立一個路徑src/hello，表示原始碼資料夾src中有一個hello應用程式目錄。

```
mkdir src
cd src
mkdir hello
cd hello
```

- 然後建立一個檔案叫做hello.go，內容為以下：

```
package main

import "fmt"

func main() {
    fmt.Printf("hello, world\n")
}
```

Start to Go

安裝Go語言

- 第一個Go程式：
- 然後在終端機中先執行`go mod init hello`，之後執行`go build`，就會編譯出一個叫做`hello.exe`的可執行檔，可以直接執行它。

```
(base) C:\Users\Alvin\go\src\hello>cd ..
(base) C:\Users\Alvin\go\src>go mod init hello
go: creating new go.mod: module hello
go: to add module requirements and sums:
    go mod tidy
(base) C:\Users\Alvin\go\src>cd hello
(base) C:\Users\Alvin\go\src\hello>go build
(base) C:\Users\Alvin\go\src\hello>dir
磁碟區 C 中的磁碟沒有標籤。
磁碟區序號: 8407-45E9

C:\Users\Alvin\go\src\hello 的目錄
2022/09/03 上午 05:01 <DIR>      .
2022/09/03 上午 05:01 <DIR>      ..
2022/09/03 上午 05:01             1,964,032 hello.exe
2022/09/03 上午 04:53             79 hello.go
                2 個檔案      1,964,111 位元組
                2 個目錄      696,896,520,192 位元組可用
```

```
(base) C:\Users\Alvin\go\src\hello>hello
hello, world
```

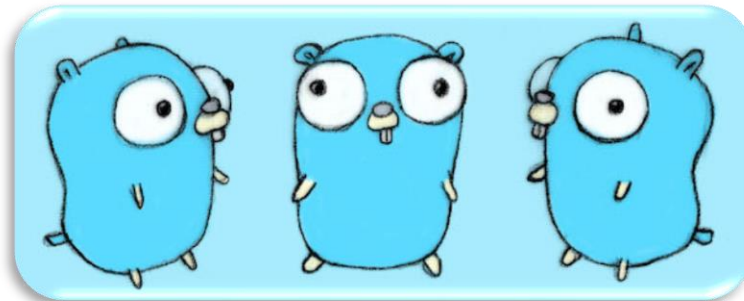
- 另外，也可以用`go run`，這會像執行腳本語言的一樣直接執行它，不會輸出編譯檔。但實際上，`go`程式碼還是會先被編譯再執行。

```
(base) C:\Users\Alvin\go\src\hello>go run hello.go
hello, world
```


Start to Go

安裝Go語言

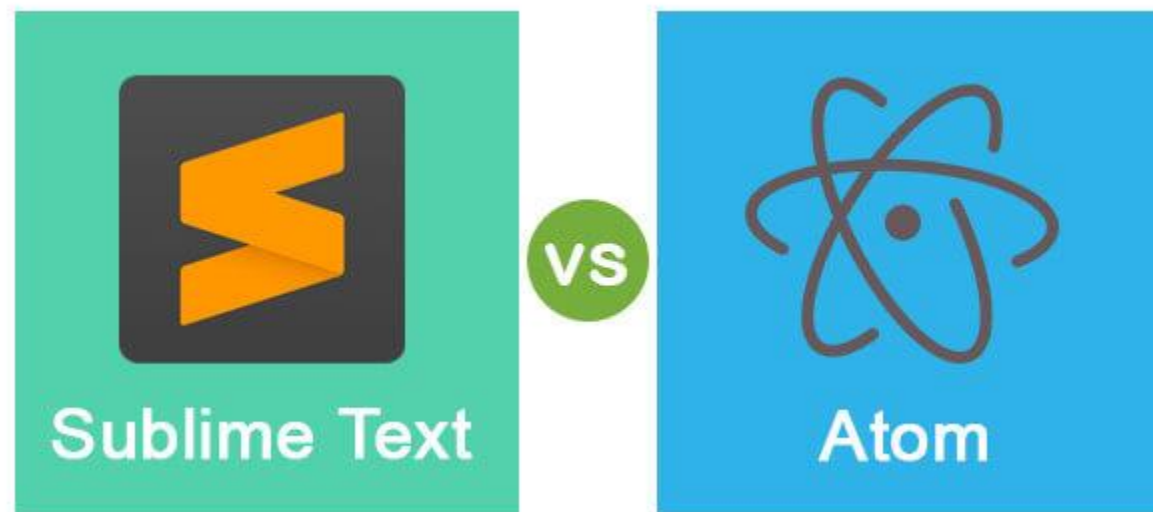
- 第一個Go程式：
 - 兩者的差別在於，主要就在於 Go 是編譯式語言。現在寫的這些 Go 的語法，最終都需要整份先轉為電腦懂的語言 (10100011 這種東西) 才能執行。
 - `go run hello.go`：就是同時編譯跟執行，所以執行此指令後，就能在 Terminal 上看到 Hello World 了。
 - `go build hello.go`：則是會產生一個執行檔，會在同個資料夾下產生一個 `hello` 的執行檔。
 - 這邊就能體現編譯式語言的好處，可以使用 `go build`，產生一個執行檔，此時可以將此 `hello_world` 執行檔，帶到不同電腦執行，而不用安裝 Go 的相關環境。



Start to Go

安裝Go語言：**Editor-Go**必先利其器

- 以下這些是免費的 Golang Editor，當然都是可以支援Windows和MacOS的
 - Atom：<https://atom.io/>
 - Sublime Text：<https://www.sublimetext.com/>



Start to Go

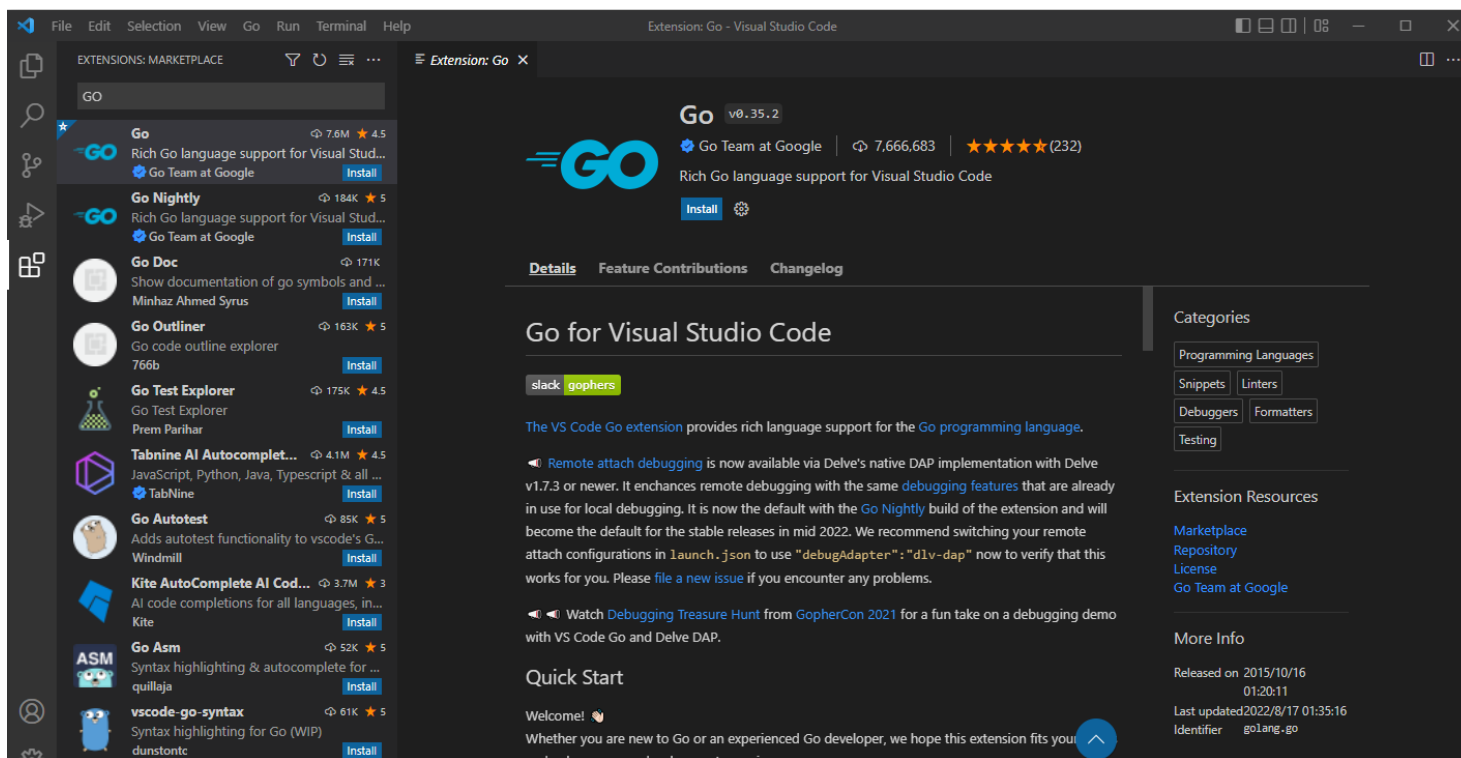
安裝Go語言：IDE(Integrated Development Environment)-Go必先利其器

- 以下這些是免費的 Golang IDE，當然都是可以支援Windows和MacOS的
 - **VScode**：<https://code.visualstudio.com/>
 - **Eclipse**：<https://www.eclipse.org/downloads/>
 - **LiteIDE**：<http://liteide.org/en/>
- 付費的IDE當然比較好用，
 - **Goland IDE**：<https://www.jetbrains.com/go/>
 - 新帳號註冊可免費使用30天。學生教育版信箱(Google教育版帳號也可以 .go.edu.tw)可以免費使用一年。一年繳199美金便可終身使用，只是一年期到以後，軟體要再更新、升版本要再繳下一個年度的費用。

Start to Go

安裝Go語言：IDE-Go必先利其器(VScode)

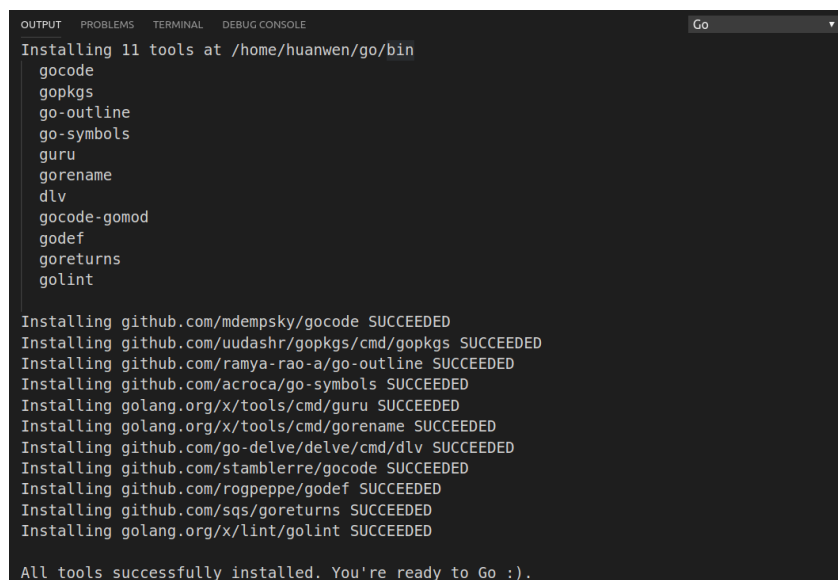
- 下載安裝完後，打開vs code，點選左側工具列的擴充(Extension)，搜尋go並且安裝它：



Start to Go

安裝Go語言：IDE-Go必先利其器(VScode)

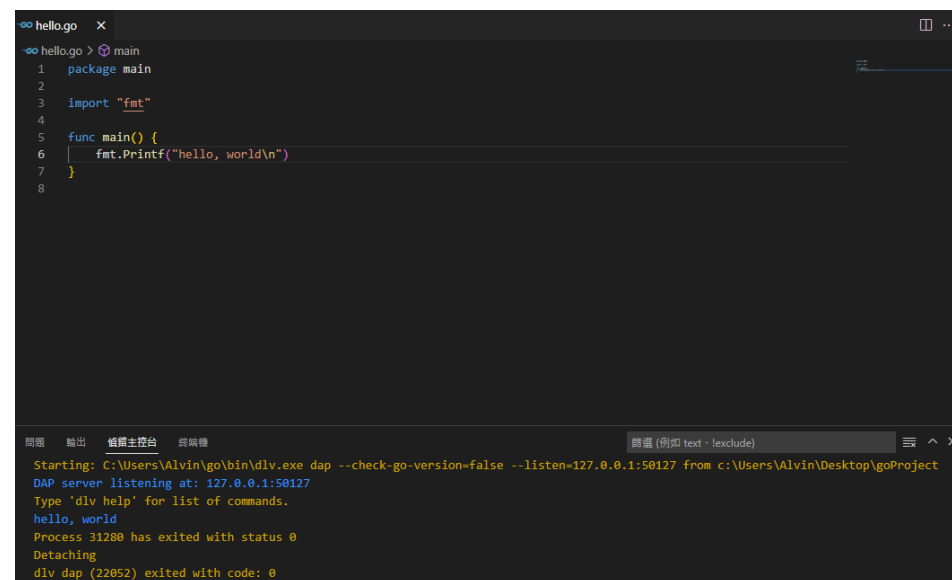
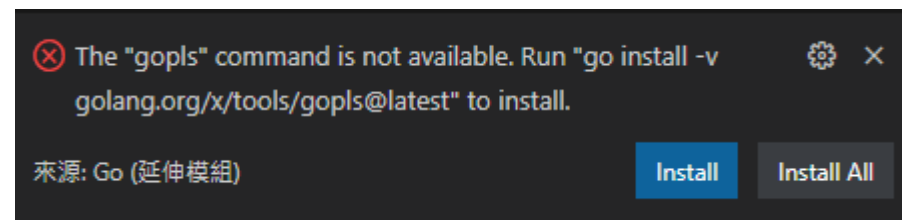
- 安裝完Go的VS擴充後，可以隨便打開一個go檔案，vs code就會在右下角提示要裝一些go的工具，安裝完就可以正常開發go程式，並且可以用vs code的中斷點偵錯。工具安裝的過程如下圖所示：



```
OUTPUT PROBLEMS TERMINAL DEBUG CONSOLE
Installing 11 tools at /home/huanwen/go/bin
gocode
gopkgs
go-outline
go-symbols
guru
gorename
dlv
gocode-gomod
godef
goreturns
golint

Installing github.com/mdempsky/gocode SUCCEEDED
Installing github.com/uudashr/gopkgs/cmd/gopkgs SUCCEEDED
Installing github.com/ramya-rao-a/go-outline SUCCEEDED
Installing github.com/acroca/go-symbols SUCCEEDED
Installing golang.org/x/tools/cmd/guru SUCCEEDED
Installing golang.org/x/tools/cmd/gorename SUCCEEDED
Installing github.com/go-delve/delve/cmd/dlv SUCCEEDED
Installing github.com/stamblerre/gocode SUCCEEDED
Installing github.com/rogpeppe/godef SUCCEEDED
Installing github.com/sqs/goreturns SUCCEEDED
Installing golang.org/x/lint/golint SUCCEEDED

All tools successfully installed. You're ready to Go :).
```



```
hello.go x
go hello.go > main
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Printf("hello, world\n")
7 }
8

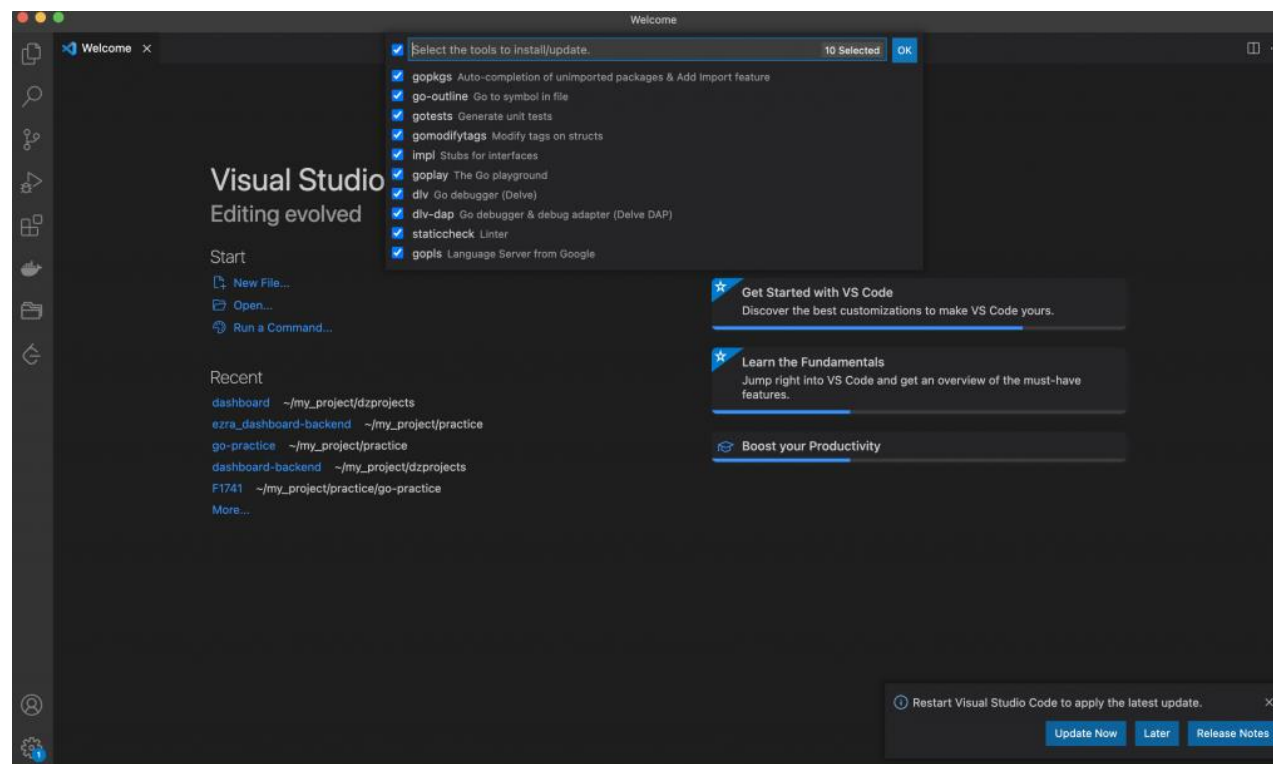
問題 輸出 偵錯主控台 偵錯欄
Starting: C:\Users\Alvin\go\bin\dlv.exe dap --check-go-version=false --listen=127.0.0.1:50127 from c:\Users\Alvin\Desktop\goProject
DAP server listening at: 127.0.0.1:50127
Type 'dlv help' for list of commands.
hello, world
Process 31280 has exited with status 0
Detaching
dlv dap (22052) exited with code: 0
```

- 按下F5可以直接執行程式，並且進入偵錯模式：

Start to Go

安裝Go語言：IDE-Go必先利其器(VScode)

- 安裝相關工具按下ctrl + shift + P(command + P)搜尋go install，勾選所有工具後按下O K。



Start to Go

安裝Go語言：IDE-Go必先利其器(VScode)

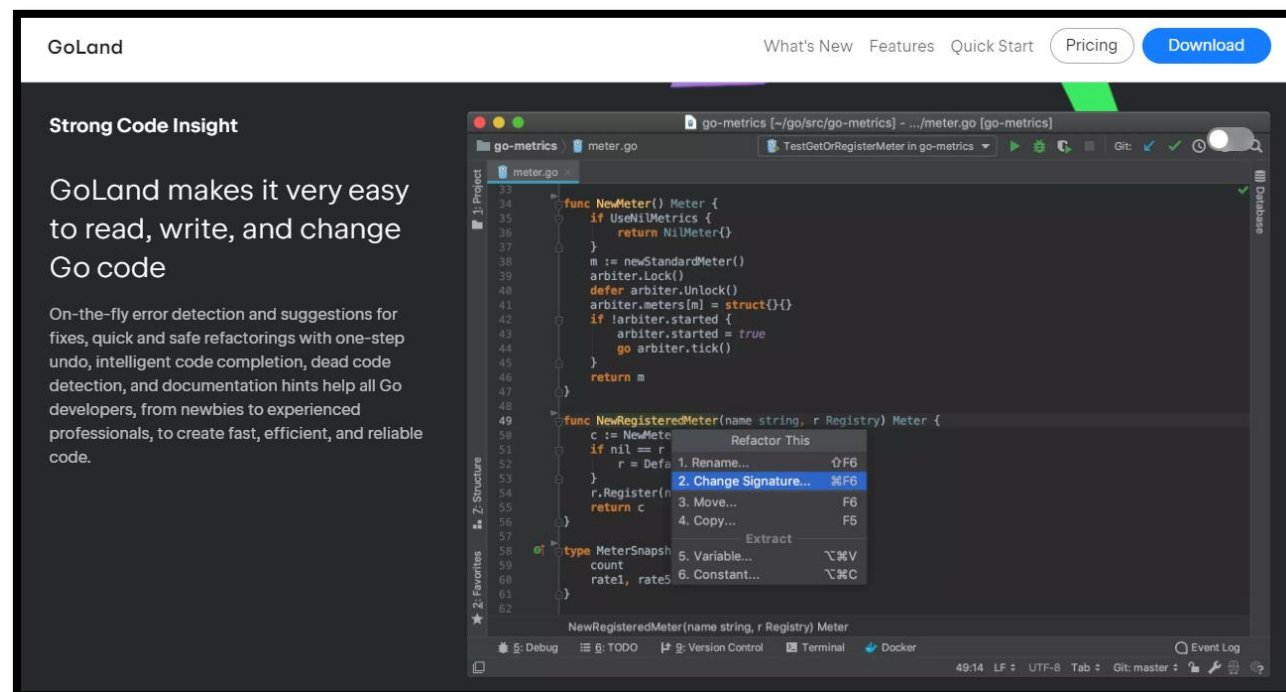
- 下載後，會方便開發程式，有一些自動排版的功能，或是 Go To Definition，詳細內容可參考：
 - <https://code.visualstudio.com/docs/languages/go>
- 接這可以開啟自動儲存的功能，就不用每次都 `command+s` 了。



Start to Go

安裝Go語言：IDE-Go必先利其器(**GoLand**)

- 另外，JetBrains這間專門做IDE(整合開發環境)的公司也有提供Go語言開發的IDE - GoLand，只是它不是免費軟體，但是有30天免費試用期，有時間可以來玩玩看。



變數、資料型態、運算子



變數、資料型態、運算子

```
package main

import "fmt"

func main() {
    fmt.Printf("hello, world\n")
}
```

程式結構

1. 所有的go程式是由package組成的，package這個字通常翻譯成套件。
2. go的主程式必須是由名為main的套件裡面的main方法(func)作為進入點。
3. package關鍵字表示這支程式是屬於哪個套件，一個套件可以有多支程式檔案，但通常是在同一個目錄下。
4. import關鍵字表示匯入其他的套件，可以透過引入的套件名呼叫其他套件的函式。
5. fmt是匯入進來的套件名，程式可以透過套件名找到套件內定義的函式並且呼叫它。

變數、資料型態、運算子

```
package main

import "fmt"

func main() {
    fmt.Printf("hello, world\n")
}
```

程式結構

■ Package

- 宣告套件(Package)，所有的Go語言檔案(.go)都必須以套件宣告起頭，透過 `package` 這個保留字來設置package名稱。
- `package` 名稱其實是 `namespace` 的概念，其目的是為了防止套件之間名稱上的衝突。一般來說，在實務上，程式語言不會只有內建的功能，一定加入第三方套件或模組，以利於所有開發者提供套件，讓其他開發者使用。
- 位於同一個目錄下的Go語言檔案，都會被視為相同套件的一部分，也就是所有檔案開頭都必須設定為相同的套件名稱。
- `package` 主要分成兩種，一種是 `executable` (可執行的)；另外一種則是 `reusable`(可重複使用的)，而 `package main` 就是指可執行的檔案。
- 因此包含 `package main` 的 Go 檔案，透過 `go build` 操作時(像是 `hello.go`)，就會產出一個 `hello.exe`，最終電腦是根據這個 `exe` 檔案去執行相關操作。

變數、資料型態、運算子

程式結構

Package

- 此時假設創造一個相加的 `package`，叫做 `calculator`，就會撰寫成這樣：
- 這樣的 `package name` 的話，就不會被 `Go` 認定為可執行檔，因此即便操作 `go build calculator.go`，也不會產生執行檔，換言之，電腦不會實際跑內部所寫的程式。
- 此時如果要印出 `calculator.go` 裡面的結果，可以寫一個可執行檔案，寫上 `package main`，並將相關套件 `import` 進來，像是以下：

```
//calculator.go
package calculator

func AddTwoNums(num1 int, num2 int) int {
    return num1 + num2
}
```

```
package main

import (
    "fmt"
    "hello/calculator"
)

func main() {
    fmt.Println(calculator.AddTwoNums(3, 4))
}
```

變數、資料型態、運算子

程式結構

▪ import

- 這就是上述提到的，引入套件，一般來說，不會從頭撰寫所有功能，這是非常耗時且不方便的。而 Go 語言的標準函式庫為開發團隊預先寫好的，提供了一些基本且常用的功能，第三方函式庫則是由 Go 語言開發團隊之外的開發者，研發並提供其他開發者使用，以補足內建以及標準函式庫的不足。

▪ import fmt

- 這個 fmt 就是一個 go 的內建標準 package(像是 python module)，當下載 Go 的時候，就一併被下載下來了，因此可以直接 import 進來使用。當未來內建標準 package 不夠用時，開始需要下載其他人的 package 來使用時，就要接著講到 GOPATH 跟 go install 和 go get 了。

```
import (  
    "errors"  
    "fmt"  
    "log"  
    "math/rand"  
    "strconv"  
    "time"  
)
```

變數、資料型態、運算子

程式結構

- 一個套件裡面可以匯入多個套件，如下：

```
package main

import (
    "fmt"
    "math"
)
// 也可以分開寫
// import "fmt"
// import "math"

func main() {
    fmt.Printf("%f\n", math.Pow(3, 2))
}
```

- 注意：匯入的多個套件之間不需要逗號(,)
- 其他比較詳細的package用法會在之後說明，目前談到的基本語法都可以在這個package main結構下實現。

變數、資料型態、運算子

程式結構

■ Main Function

- 每個 Go 語言下的專案基本上都會有一個主程式(供給給開發者使用的套件可能不需要)主程式裡的程式通常為最核心的部分，但在一開始寫程式時，通常會把全部的東西都塞在裡面，這是不方便閱讀以及除錯的，所以在累積一段經驗後，會將部分程式碼切出來寫成另一個函數，甚至分成其他package。
- 在 func main 外面，寫下其他的 function，再一併在 main 內執行。如下示範：

```
package main

import (
    "fmt"
)

func addTwoNums(num1 int , num2 int )int {
    return num1+num2
}

func main() {
    fmt.Println(addTwoNums(3,4))
}
```

變數、資料型態、運算子

程式結構

- 語法斷句
 - 在Go中，語法斷句需要分號; 但是不用寫(硬要寫也是可以)。
 - 因為它會自動產生，可以省鍵盤軸心壽命。

```
a := 10;  
b := 20;;  
fmt.Println(a, b);;;;
```


變數、資料型態、運算子

程式結構

- Comments

- `'//'` 之後的同一行文字視為註解。註解不為真正的程式碼，一般來說，註解是用來解釋某段程式碼的用途。每個程式設計師，在寫完程式碼後，過了一段時間後，必定也會有忘記某段程式碼的用途，或是方便自己查詢自己要的程式碼，也是方便其他開發者，能夠清楚明白程式碼的用意，所以寫註解也是相當重要的一門學問。

- 兩種形式為 `//` (單行註解) 及 `/* */` (多行註解)

- 另外補上字串連接方法：`fmt.Println("Google" + "GCP")`

- 也可以使用`print`多個字串的方式處理：`fmt.Println("String", "Concatenation!!")`

- 也可以使用`Sprintf`來處理：

```
str := fmt.Sprintf("%s%s", "Google", "GCP")  
  
fmt.Println(str)
```

變數、資料型態、運算子

資料型態

- 數值：Golang的數值型態主要分成兩種：**整數與浮點數**
 - 整數的部份：整數代表不帶有小數點的數值(.00也不算是整數)

型態	說明	範例
uint	unsigned integer，意味沒有符號的整數	0,1,2,3
int	integer，整數	-3,-2,0,1

- 在C語言有sizeof可以取得變數佔用的記憶體大小，而在go語言也可以透過unsafe.Sizeof()取得記憶體大小，以下範例：

```
// 將 1 左移 7 位元再 - 1 (不 - 1 會錯誤)
var a int8 = 1<<7 - 1
fmt.Printf("int8 長度= %d byte, 最大值= %d\n", unsafe.Sizeof(a), a)

// 將 1 左移 8 位元再 - 1 (不 - 1 會錯誤)
var b uint8 = 1<<8 - 1
fmt.Printf("uint8 長度= %d byte, 最大值= %d\n", unsafe.Sizeof(b), b)
```

```
int8 長度= 1 byte, 最大值= 127
uint8 長度= 1 byte, 最大值= 255
```

```
uint8    the set of all unsigned 8-bit integers (0 to 255)
uint16   the set of all unsigned 16-bit integers (0 to 65535)
uint32   the set of all unsigned 32-bit integers (0 to 4294967295)
uint64   the set of all unsigned 64-bit integers (0 to 18446744073709551615)

int8     the set of all signed 8-bit integers (-128 to 127)
int16    the set of all signed 16-bit integers (-32768 to 32767)
int32    the set of all signed 32-bit integers (-2147483648 to 2147483647)
int64    the set of all signed 64-bit integers (-9223372036854775808 to 9223372036854775807)

float32  the set of all IEEE-754 32-bit floating-point numbers
float64  the set of all IEEE-754 64-bit floating-point numbers

complex64 the set of all complex numbers with float32 real and imaginary parts
complex128 the set of all complex numbers with float64 real and imaginary parts

byte     alias for uint8
rune     alias for int32
```

變數、資料型態、運算子

資料型態

- 數值：Golang的數值型態主要分成兩種：**整數與浮點數**

- 整數的部份：整數代表不帶有小數點的數值(.00也不算是整數)

- 以「電腦科學」的角度來出發，電腦是怎麼表達數字的，大家都知道電腦內部是由 0 和 1 來運作的，那怎麼表示整數呢？

- 00000000
- 00000001
- 00000010
- 00000011
- 00000100
- ...

- 其中一個存取 0 或 1 的單元稱作「位元」(bit)，而每個例子中都有 8bits，根據排列組合計數原理，一共可以有 256 排列：
- 在 Golang 中稱這個叫int8，可以用來表示(-128~127)

uint8	the set of all unsigned 8-bit integers (0 to 255)
uint16	the set of all unsigned 16-bit integers (0 to 65535)
uint32	the set of all unsigned 32-bit integers (0 to 4294967295)
uint64	the set of all unsigned 64-bit integers (0 to 18446744073709551615)
int8	the set of all signed 8-bit integers (-128 to 127)
int16	the set of all signed 16-bit integers (-32768 to 32767)
int32	the set of all signed 32-bit integers (-2147483648 to 2147483647)
int64	the set of all signed 64-bit integers (-9223372036854775808 to 9223372036854775807)
float32	the set of all IEEE-754 32-bit floating-point numbers
float64	the set of all IEEE-754 64-bit floating-point numbers
complex64	the set of all complex numbers with float32 real and imaginary parts
complex128	the set of all complex numbers with float64 real and imaginary parts
byte	alias for uint8
rune	alias for int32

```
10000000 (-128)
10000001 (-127)
10000010 (-126)
...
11111111 (-1)
00000000 (0)
00000001 (1)
00000010 (2)
...
01111110 (126)
01111111 (127)
```

變數、資料型態、運算子

資料型態

- 數值：Golang的數值型態主要分成兩種：**整數與浮點數**
 - 整數的部份：整數代表不帶有小數點的數值(.00也不算是整數)
 - 有時並不會用到負號，這時記作 **uint8** 可以表示(0~255)

```
00000000 (0)
00000001 (1)
00000010 (2)
...
10000000 (128)
10000001 (129)
...
11111111 (255)
```

- **uint** 只能是正數，**int** 可以有負數
- **int** 表示負數的存法在電腦科學中稱為「二補數」

uint8	the set of all unsigned 8-bit integers (0 to 255)
uint16	the set of all unsigned 16-bit integers (0 to 65535)
uint32	the set of all unsigned 32-bit integers (0 to 4294967295)
uint64	the set of all unsigned 64-bit integers (0 to 18446744073709551615)
int8	the set of all signed 8-bit integers (-128 to 127)
int16	the set of all signed 16-bit integers (-32768 to 32767)
int32	the set of all signed 32-bit integers (-2147483648 to 2147483647)
int64	the set of all signed 64-bit integers (-9223372036854775808 to 9223372036854775807)
float32	the set of all IEEE-754 32-bit floating-point numbers
float64	the set of all IEEE-754 64-bit floating-point numbers
complex64	the set of all complex numbers with float32 real and imaginary parts
complex128	the set of all complex numbers with float64 real and imaginary parts
byte	alias for uint8
rune	alias for int32

變數、資料型態、運算子

資料型態

uint8	the set of all unsigned 8-bit integers (0 to 255)
uint16	the set of all unsigned 16-bit integers (0 to 65535)
uint32	the set of all unsigned 32-bit integers (0 to 4294967295)
uint64	the set of all unsigned 64-bit integers (0 to 18446744073709551615)
int8	the set of all signed 8-bit integers (-128 to 127)
int16	the set of all signed 16-bit integers (-32768 to 32767)
int32	the set of all signed 32-bit integers (-2147483648 to 2147483647)
int64	the set of all signed 64-bit integers (-9223372036854775808 to 9223372036854775807)
float32	the set of all IEEE-754 32-bit floating-point numbers
float64	the set of all IEEE-754 64-bit floating-point numbers
complex64	the set of all complex numbers with float32 real and imaginary parts
complex128	the set of all complex numbers with float64 real and imaginary parts
byte	alias for uint8
rune	alias for int32

- 數值：Golang的數值型態主要分成兩種：整數與浮點數
 - 浮點數的部份：浮點數囊括任何帶有小數點的數值(.00也算在浮點數唷)
 - 電腦都是按照 IEEE754 來處理浮點數的，這部分比較複雜，簡單來說就是將一串浮點數拆成科學記號來表示，在 Golang 中一共有兩種小數的宣告方法，分別為：
 - float32
 - float64
 - 因為對電腦來說，考慮到效能，無法完美處理小數，所以有兩種提供選擇。其中float64花了2倍的空間所以存的比 float32 還要精準，通常會稱 float64 為雙精度浮點數
 - 雙精度符點數
 - 有些語言會以 float 代表 float32, double 代表 float64，但是如果是採用 32, 64 這種表示法，就可以很清礎的了解該變數到底占用了多少空間。

變數、資料型態、運算子

資料型態

- 數值：Golang的數值型態主要分成兩種：**整數與浮點數**
 - 浮點數的部份：浮點數囊括任何帶有小數點的數值(.00也算在浮點數唷)
 - 用`unsafe.Sizeof()`看看佔用的記憶體大小：

```
var a float32 = math.MaxFloat32
fmt.Printf("float32 長度= %d byte\n最大值(科學記號法)= %e\n最大值(十進制)= %f\n", unsafe.Sizeof(a), a, a)

var b float64 = math.MaxFloat64
fmt.Printf("float64 長度= %d byte\n最大值(科學記號法)= %e\n最大值(十進制)= %f\n", unsafe.Sizeof(b), b, b)
```

- 範例直接拿`math`套件的`float`最大值來用，分別印出長度與大小值，而浮點數值除了使用一般十進制(`%f`)印出，也可以用科學記數法(`%e`)印出。
- 另外，go語言還有提供兩個複數型別`complex64`, `complex128`，但是複數在應用上是滿少見的，

uint8	the set of all unsigned 8-bit integers (0 to 255)
uint16	the set of all unsigned 16-bit integers (0 to 65535)
uint32	the set of all unsigned 32-bit integers (0 to 4294967295)
uint64	the set of all unsigned 64-bit integers (0 to 18446744073709551615)
int8	the set of all signed 8-bit integers (-128 to 127)
int16	the set of all signed 16-bit integers (-32768 to 32767)
int32	the set of all signed 32-bit integers (-2147483648 to 2147483647)
int64	the set of all signed 64-bit integers (-9223372036854775808 to 9223372036854775807)
float32	the set of all IEEE-754 32-bit floating-point numbers
float64	the set of all IEEE-754 64-bit floating-point numbers
complex64	the set of all complex numbers with float32 real and imaginary parts
complex128	the set of all complex numbers with float64 real and imaginary parts
byte	alias for uint8
rune	alias for int32

```
float32 長度= 4 byte
最大值(科學記號法)= 3.402823e+38
最大值(十進制)= 340282346638528859811704183484516925440.000000
```

```
float64 長度= 8 byte
最大值(科學記號法)= 1.797693e+308
最大值(十進制)= 179769313486231570814527423731704356798070567525844996598917476
803157260780028538760589558632766878171540458953514382464234321326889464182768
467546703537516986049910576551282076245490090389328944075868508455133942304583
236903222948165808559332123348274797826204144723168738177180919299881250404026
184124858368.000000
```

變數、資料型態、運算子

資料型態

uint8	the set of all unsigned 8-bit integers (0 to 255)
uint16	the set of all unsigned 16-bit integers (0 to 65535)
uint32	the set of all unsigned 32-bit integers (0 to 4294967295)
uint64	the set of all unsigned 64-bit integers (0 to 18446744073709551615)
int8	the set of all signed 8-bit integers (-128 to 127)
int16	the set of all signed 16-bit integers (-32768 to 32767)
int32	the set of all signed 32-bit integers (-2147483648 to 2147483647)
int64	the set of all signed 64-bit integers (-9223372036854775808 to 9223372036854775807)
float32	the set of all IEEE-754 32-bit floating-point numbers
float64	the set of all IEEE-754 64-bit floating-point numbers
complex64	the set of all complex numbers with float32 real and imaginary parts
complex128	the set of all complex numbers with float64 real and imaginary parts
byte	alias for uint8
rune	alias for int32

- 數值：Golang的數值型態主要分成兩種：整數與浮點數
 - 浮點數的部份：浮點數囊括任何帶有小數點的數值(.00也算在浮點數唷)
 - 在這邊另外介紹一個東西叫做強轉型，不同型別的資料可以相互轉換，但有些時候會錯誤或是損失精準度，go語言的轉型方式比較特別：[型別] ([表達式])
 - 如果float32跟float64要做運算的話，正常情況這兩個型別是無法直接做運算，所以float32要做強轉，在這邊建議，一般是建議小type轉成大type，而不要大type轉小type，以免造成數值失真，範例如下：

```
package main

import (
    "fmt"
)

func main() {
    var (
        a float64 = 1
        b float32 = 1
        c float64 = 1.2
    )
    d := a + float64(b)
    fmt.Println(d)
    fmt.Println(c + d)
}
```

變數、資料型態、運算子

資料型態

- 字串

型態	範例
string	"This is a book", "Hello World!"

- 字串來說非常容易，就是以雙引號包起來的都叫做字串。
- 簡易的做變數字串串接：

```
package main

import (
    "fmt"
)

func main() {
    var (
        a string = "Hello"
        b string = "World"
    )
    d := a+b
    fmt.Println(d)
}
```

這種字串串接的方法不是 **best practice**，
原因在於這種方式會額外產生一塊記憶體空間。

變數、資料型態、運算子

資料型態

■ 字串

- 用反引號(`)定義多行文字：

```
s := `IronMan,  
    his name is Tony Stark,  
    and he is my favorite hero.  
`  
fmt.Println(s)
```

```
IronMan,  
    his name is Tony Stark,  
    and he is my favorite hero.
```

- 因為go語言會把、符號之間的所有符號(包含空白or換行)都當成字串內容，所以上面印出結果才會沒有對齊。
- 比較特別的是，go語言沒有字元的基本型別，反而有字串型別。go語言的字元是用整數來表達，雖然C語言的字元也可以轉換成整數(編碼)，但仍然有原生的字元型別。以下舉個範例，可以用格式化輸出(%T)變數的型別：

```
var ascii byte = 'I'  
fmt.Printf("%d %T \n", ascii, ascii)  
  
var utf8 rune = '鋼'  
fmt.Printf("%d %T \n", utf8, utf8)
```

```
73 uint8  
37628 int32
```

從程式碼與輸出結果可以得知，go字元可以用byte或rune表達，視字元的編碼長度而定。然後，會發現byte和rune都是整數型別的一種。

變數、資料型態、運算子

資料型態

- byte
 - Byte用法非常的多，通常都會搭配Slice[]：

```
func main() {  
    var b = []byte("0")  
    fmt.Println(b)  
  
    b = []byte("爆肝工程師的異世界安魂曲")  
    fmt.Println(b)  
}  
  
/* result:  
[48]  
[231 136 134 232 130 157 229 183 165 231 168 139 229 184 171 231 154 132 231 1  
49 176 228 184 150 231 149 140 229 174 137 233 173 130 230 155 178]  
*/
```

變數、資料型態、運算子

資料型態

▪ rune

- rune意思是符號，簡單理解為UTF-8中的一個字符(元)。
- Go語言預設使用UTF-8編碼，而UTF-8編碼中的每個字元長度不是固定的，以8bit為基本單位長度，範圍為1 ~ 4個字節 (1 byte ~ 4 bytes)。

```
func main() {  
    var r = []rune("0")  
    fmt.Println(r)  
  
    r = []rune("爆肝工程師的異世界安魂曲")  
    fmt.Println(r)  
}  
  
/* result:  
[48]  
[29190 32925 24037 31243 24107 30340 30064 19990 30028 23433 39746 26354]  
*/
```

像常用到的ASCII中的英文、數字在UTF-8中只佔一個 byte(8bits)，而中文、韓文、日文一個字符則用到2~3 bytes不等。

變數、資料型態、運算子

資料型態

- rune

- 要注意下面這樣的狀況：長度到底是算符號數還是算byte數？第一次看到會不曉得rune的人看到程式碼的輸出，會覺得奇怪：

```
func main() {  
    a := "Hi,世界"  
    fmt.Println(a)  
    n := 0  
    for range a{  
        n++  
    }  
    fmt.Println(len(a))  
    fmt.Println(n)  
}  
  
/* result:  
Hi,世界  
9  
5  
*/
```

可以看到，以for range迭代印出來的長度，就是老老實實的以bytes來計算，而len()印出來的長度是符號數，每個文字都只算一個。

變數、資料型態、運算子

資料型態

- 布林值：布林值就是true or false的數值

```
package main

import (
    "fmt"
)

func main() {
    var (
        a bool = true
        b bool = false
    )

    fmt.Println(a && b)
    fmt.Println(a || b)
}
```

變數、資料型態、運算子

資料型態

- 型態別名(Type Alias)

- go可以對任何一個型別定義別名，而別名就像是一個外號，本質上還是同一型別。別名只存在於編譯期間，進入執行期間就會回到原始型別。

- 格式：`type [別名] = [型別]`

```
// 定義一個 int 的別名叫做 myInt  
type myInt = int
```

- 而go語言本身也有內建的型別別名：

```
type byte = uint8  
type rune = int32
```

- 從上面可以看到其實byte是uint8、rune是int32，通過定義別名方式獲得新的型別，這樣可以用於一些特別的需求，提高程式碼可讀性以及型別安全。

變數、資料型態、運算子

資料型態

- 型態別名(Type Alias)
 - 用格式化輸出(%T)型別，來看看別名的原始型別：

```
type IntAlias = int  
  
var a byte  
fmt.Printf("%T \n", a)  
  
var b rune  
fmt.Printf("%T \n", b)  
  
var c IntAlias  
fmt.Printf("%T \n", c)
```

```
uint8  
int32  
int
```

變數、資料型態、運算子

資料型態

- 型態別名(Type Alias)
 - 因此，可以看到在程式的執行期間，別名就會被改成原始型別。另外要注意，**千萬不要少加一個等號**，意義完全不同，會變成定義一個新的型別：

```
type NewInt int

var a NewInt

fmt.Printf("%T \n", a)
// 印出: main.NewInt
```

- 因為type關鍵字本身就是用於定義型別，像是結構、介面也是要用type定義，別名的語法必須加上等號(=)來區分這是在定義別名，而不是新的型別。

變數、資料型態、運算子

變數宣告

- 變數宣告了就要使用，否則會出現 `variable declared and not used` 的錯誤。除非變數的名稱為 `_`。

```
var Name Type = Expression
```

```
var a int = 16
```

- 可省略型別 `Type`，由 `Expression` 決定 `Type`
- 可省略表達式 `Expression`，變數值為 `Type` 的初始值(零值)
- 初始值(零值)：0, false, "", nil 等等

```
func main() {  
    var a = 16  
    var b int  
    var _ = 10  
    fmt.Println(a, b)  
}  
// result:  
// 16 0
```

變數、資料型態、運算子

變數宣告

- 除了整數、字串、浮點數這些常見的基本資料型別之外，go也支援函式變數，可以把方法當成變數或參數在程式之中傳遞。其他還有比較特殊的型別，像是結構、介面、通道等。

```
// 宣告一個名叫a的整數變數
var a int

// 宣告一個名叫b的字串變數
var b string

// 宣告一個名叫c的浮點數陣列
var c []float32

// 宣告一個名叫d的方法變數並且回傳bool型別
var d func() bool
```

```
// 若型別相同，可以這樣寫
var a, b, c int

// 若型別不同，可以這樣寫
var (
    a int
    b string
    c []float32
    d func() bool
)
```

- 另外也有多重寫法，用一個var關鍵字宣告多個變數：

變數、資料型態、運算子

變數宣告

- 賦值(assignment)

- go變數賦值方式與大部分的程式語言相同，可以在宣告式右邊加上 = 與初始值:
- 格式：`var [變數名稱] [變數型別] = [初始值]`

```
var year int = 2020  
var name string = "Iron man"
```

- 由於初始值2020本身就是整數，所以在宣告變數時可以省略int型別，這是因為go變數有支援型別推導的功能，和C#的var一樣，從右側的初始值或是一個方法呼叫的回傳值推導出變數型別：

```
var year = 2020    // int 型別  
var name = "Iron man"  // string 型別  
  
// 多重寫法  
var price, name = 2020, "Iron man"
```

變數、資料型態、運算子

變數宣告

- 賦值(assignment)

```
var a, b = 1, 2

// 傳統swap寫法
var temp int
temp = a
a = b
b = temp

// 多重賦值swap寫法
a, b = b, a
```

- 多重賦值寫法對於變數交換(`swap`)有很大的幫助：
 - 程式碼從四行降到一行，而且不用宣告`temp`。

變數、資料型態、運算子

變數宣告

- 匿名佔位符(anonymous placeholder)

- placeholder的意思就是用一個符號(`_`)佔據原本放變數的位置，目的就是想要忽略等號右側的賦值。

- 格式: [變數名稱], `_` = [第一個值], [第二個值]

```
// 單一賦值雖然不會編譯錯誤，但這一行是沒有意義的
_ = "Iron man"

// 多重賦值下才會有意義，可以忽略不想要的值
var name string
name, _ = "Iron man", "Dr.strange"
```

- 通常在等號的右側是方法呼叫時，匿名佔位符會更常被用到，可以只接受部份回傳值。

```
// go方法可以有多個回傳值
func getHeroName() (string, string) {
    return "Iron man", "Dr.strange"
}

func main() {
    var name, _ = getHeroName()
    fmt.Println("I like " + name)
}
```

變數、資料型態、運算子

變數宣告(作用範圍)

- 通常一個變數宣告的位置會影響它的使用範圍，這個稱為作用域(scope)。
- 主要分為**全域變數和區域變數**，範例如下：

```
package main

var a int // 全域變數

func main() {
    var b string // 區域變數
}
```

```
package main

import "fmt"

var a int // 全域變數

func main() {
    var b string // 區域變數
    fmt.Println(b)
}
```

- 全域變數a可以在整個package main裡使用，但區域變數b就只能在function main裡使用。
- 此外，如果執行左上的程式，會出現以下錯誤訊息：`./hello.go:8:6: b declared and not used`
- go語言的區域變數被宣告之後，如果沒使用它就算是編譯錯誤，改成右上程式碼就會編譯成功了。

變數、資料型態、運算子

變數宣告(作用範圍)

- 所有變數都有其運作範圍(或稱層級、作用域)。最上層的範圍是套件(package)範圍。
- 變數範圍的上下層關係在編譯程式時就決定好，不是等到執行階段才決定。當某段程式碼存取某個變數時Go語言會檢查該程式碼的作用範圍。
- 範圍內找不到該名稱，就會往上一層範圍找，一直找到最頂層的套件範圍為止。途中找到同名變數，Go就會停止搜尋，並使用那個變數，但若到頂端還是找不到，就會拋出錯誤。

```
package main

import "fmt"

var level = "pkg"

func main() {
    fmt.Println("Main start :", level) //main()層級
    if true {
        fmt.Println("Block start :", level) //底下if層級
        funcA()
    }
}

func funcA() {
    fmt.Println("funcA start :", level) //funcA()層級
}
```

變數、資料型態、運算子

變數宣告(變數遮蔽)

- 因為Go語言一找到變數，Go就會停止搜尋，並使用那個變數，所以導致**遮蔽現象**

```
package main

import "fmt"

var level = "pkg"

func main() {
    fmt.Println("Main start :", level)

    level := 42
    if true {
        fmt.Println("Block start :", level)
        funcA()
    }
    fmt.Println("Main end   :", level)
}

func funcA() {
    fmt.Println("funcA start :", level)
}
```

```
Main start : pkg
Block start : 42
funcA start : pkg
Main end   : 42
```

由以上可得出，子範圍的level變數『遮蔽』了套件範圍的level變數。其中特別注意的是，當呼叫funcA()時，Go語言動用了靜態範圍解析，不會管funcA()在哪裏被呼叫，因此funcA()仍是套件層級的level變數！

變數、資料型態、運算子

變數宣告

- 另外，在go語言中命名開頭的大小寫是有用處的，在函式以外的地方宣告變數、方法、結構等等，只要是名稱開頭是大寫的，表示可以被匯出(**exported**)，意思就是可被其他的套件(**package**)使用；反之名稱開頭是小寫的話，就只有目前的套件可以使用。

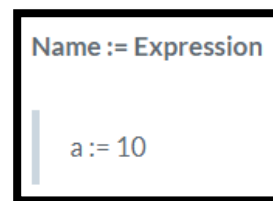
```
// 僅供目前的包使用  
var a int  
  
// 匯出給外部的包使用  
var A int
```

- 匯出有點像是Java, C#的public/private，可以限制外部程式存取元件內的資源，在其他語言中JavaScript也是有import/export特性。

變數、資料型態、運算子

變數宣告

- 短變數宣告(Short variable declarations)(用:= 代替var)



```
Name := Expression  
  
a := 10
```

- go變數還有一個更精簡的寫法，可以同時宣告與初始化，稱為短變數宣告(short variable declarations)，使用 := 運算子，不是賦值時用的 = 運算子。

```
year := 2020  
name := "Iron man"  
  
// 多重寫法  
price, name := 2020, "Iron man"
```

- **宣告declare(:=) 跟 指派assign(=)是不同的**，一次宣告多個變數，若已宣告過的變數會自動變成assign效果

```
func main() {  
    var b int = 123  
    a, b := 100, 99  
    c, b := 0, 1  
    fmt.Println(a, b, c)  
}  
  
// result:  
// 100 1 0
```

- 短變數宣告 (:=) 方法簡便，常用在宣告初始化大量、生命週期短的区域變數 (就是迴圈內的i, j)
- 反之，var 則常用在明確設定型別、生命週期較長的變數。

變數、資料型態、運算子

變數宣告

- 注意：宣告區域變數

```
package main

import "fmt"

var a = "Hello!"

func main() {
    b := 10
    fmt.Println(&a, a, &b, b)

    a, b := 100, 99
    fmt.Println(&a, a, &b, b)
}

/* result:
0x54dc50 Hello! 0xc00002c008 10
0xc00002c048 100 0xc00002c008 99
*/
```

- 只要有 var或:= 出現，就是宣告新的變數、挪用新的記憶體空間？

變數、資料型態、運算子

變數宣告

- 注意：宣告區域變數

```
package main

import "fmt"

var i = 123 // 全域變數

func main() {
    fmt.Println(&i)

    i = 123
    fmt.Println(&i)

    i := 123 // 全新的・func區塊內的區域變數
    fmt.Println(&i)

    i, j := 123, 100
    fmt.Println(&i, j) // 奇怪？i沒有變新的啊
}

/* 運行結果
0x11662a0
0x11662a0
0xc000b2008
0xc000b2008 100
*/
```

- 只要有 `var`或`:=` 出現，就會挪用全新的記憶體空間嗎？答案是：**不一定**。
- 程式會去查找目前最小單位的區塊內 是否已經有宣告過的變數，沒有的話才會宣告新的。

變數、資料型態、運算子

變數宣告

■ 注意：宣告區域變數

```
package main

import "fmt"

var x = 123 // 全域變數

func main() {
    fmt.Println(&x)

    var x = 123 // 在`func`區塊，全新。
    fmt.Println(&x)

    // var x, y = 123, 100 // 因為x已經宣告過了，不能再用`var`，但以下卻可以
    x, y := 123, 100 // 在`func`區塊，此時x是上面宣告過的區域變數，而y是新的
    fmt.Println(&x, &y)

    if true {
        var x, y = 123, 100 // 在新區域`if`區塊，此時x跟y都是全新的。
        // 可以換成短變數宣告。

        fmt.Println(&x, &y)
    }

    fmt.Println(&x) // 脫離if區域，回到`func`區塊
}

/* 運行結果
0x11662a0
0xc000016080
0xc000016080 0xc000016088
0xc000016090 0xc000016098
0xc000016080
*/
```

- 這個變數到底是不是全新的？
 - 看到有 **var** 會是全新的。
 - 但看到 **:=** 卻是不一定，短變數宣告可以伴隨著未宣告的變數一起出現。
 - 之後會常常看到下面這種用法：

```
connect, err := ...(server)
```
 - 或者：

```
result, err := ...
```
 - 此時 **:=** 左邊的兩個變數，都會是區域變數，不是外面宣告的全域變數。
 - 另外：宣告全域變數只能用 **var**，無法用 **:=**

變數、資料型態、運算子

變數宣告

- 保留字
 - 這些字用來保持Golang的運作，不能當變數名稱

break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

變數、資料型態、運算子

運算子

- 賦值語法：

```
package main
import "fmt"

func main(){
    var num int = 10
    num += 10 // 意思同 num = num + 10
    fmt.Println(num)
}
```

- 可能需要一些時間習慣：

```
num += 10 // num = num + 10
num -= 10 // num = num - 10
num *= 10 // num = num * 10
num /= 10 // num = num / 10
num %= 10 // num = num % 10
```

變數、資料型態、運算子

運算子

- 通常加減乘除是一定要有的，以整數來舉例：

```
package main
import "fmt"
func main(){
    n := 10
    m := 20
    fmt.Println(n + m)
    fmt.Println(n - m)
    fmt.Println(n * m)
    fmt.Println(m / n)
    fmt.Println(n / m)
}
```

執行結果

30
-10
200
2
0

- 在絕大多數的語言中，整數對整數運算出來的都是整數，即使是除法也不例外， n / m 是 $10/20$ 在數學中應為 0.5 ，但在電腦科學中，會直接當作 **0**

變數、資料型態、運算子

運算子

- 取餘數
 - 另外，在電腦科學中，有一種計算方式叫作「取餘數」，用「%」當做記號，舉個例子：

30 % 2 會是 0 因為 $30/2 = 15...0$
29 % 3 會是 2 因為 $29/3 = 9...2$
23 % 5 會是 3 因為 $23/5 = 4...3$

```
package main
import "fmt"
func main(){
    fmt.Println(30 % 2)
    fmt.Println(29 % 3)
    fmt.Println(23 % 5)
}
```

執行結果

0
2
3

變數、資料型態、運算子

運算子

- 遞增遞減

```
package main
import "fmt"

func main(){
    var num int = 10
    num++    // 同 num = num + 1
    fmt.Println(num)
    num--    // 同 num = num - 1
    fmt.Println(num)
}
```

執行結果：

11

10

- 可以很明顯的知道這裡指的 ++ 就是指將 num 遞增，而 -- 就是將 num 遞減。
- 然而，如果是學過其他語言的就會問那 ++num 呢？不好意思，被廢掉了。還有那 fmt.Println(num++) 呢？不好意思，也不支援，所以也沒有先加後加的問題了。也因此遞增遞減運算符在 Golang 中變的比較沒那麼複雜

變數、資料型態、運算子

運算子

■ 浮點數運算

- 浮點數運算基本上和整數差不多，但要注意一下，**浮點數可以直接和整數(單純數字)做運算，但不能和整數變數(英文)做運算**。因為和純數字時Go會直接轉成浮點數。另一方面在Golang中取餘數並不適用於浮點數。

```
package main
import "fmt"
func main(){
    m := 2.5
    m = m / 2
    fmt.Println(m)
}
```

執行結果
1.25

```
package main
import "fmt"
func main(){
    m := 2.5
    n := 2
    m = m / n
    fmt.Println(m)
}
```

```
Exec-in-cmd
# command-line-arguments
.\lesson02.go:6:11: invalid operation: m / n
(mismatched types float64 and int)
```

- 如果真的這麼想要計算要怎麼辦？那麼就轉換型別吧！把浮點數轉成整數或著把整數轉成浮點數

變數、資料型態、運算子

運算子

- 布林運算

- NOT：也就是讓 true 變 false，false 變 true，只要在布林變數前加上 ! 就可以實現

```
package main
import "fmt"
func main(){
    // 利用 := 快速宣告
    a := true
    fmt.Println(!a)
    fmt.Printf("%t", !a)
}
```

執行結果：
false
false

- 布林變數可以直接印出，若要用 Printf 來印出，可以用 %t，t 是指 true，因為 %b 的 b 被拿去當 binary 印二進位了，所以要印出布林變數要用 %t

變數、資料型態、運算子

運算子

- 布林運算

- AND：當兩件事情**同時**是 `true` 時，則 AND 的結果也是 `true`，AND 可以用中文想成「而且」，在 Golang 及大多數程式語言中(`python`除外)，用 **`&&`** 來表示「而且」，舉個例子：

```
package main
import "fmt"
func main(){
    fmt.Println(false && false)
    fmt.Println(false && true)
    fmt.Println(true && false)
    fmt.Println(true && true)
}
```

執行結果：

false
false
false
true

變數、資料型態、運算子

運算子

- 布林運算

- OR：當兩件事情**只要有一件**是 true 時，則 OR 的結果就會 true，OR 可以用中文想成「或者」，在 Golang 及大多數程式語言中(python除外)，用 **||** 來表示「或者」，舉個例子：

```
package main
import "fmt"
func main(){
    fmt.Println(false || false)
    fmt.Println(false || true)
    fmt.Println(true || false)
    fmt.Println(true || true)
}
```

執行結果：

false
true
true
true

變數、資料型態、運算子

運算子

- 位元運算(and, or, xor, bit clear)
 - 位元運算指的是針對二進位的數所設計的運算元，常見的有：
 - & (and), | (or), ^ (xor), &^ (bit clear), >> (right shift), << (left shift)
 - 以下示範，用 0b 開頭來表示二進位，比如 0b10 代表 2, 0b100 代表 4：

```
package main
import "fmt"

func main(){
    // 只有左右 bit 都為 1，結果才會是 1
    fmt.Printf("%b\n", 0b0 & 0b0)    // 0
    fmt.Printf("%b\n", 0b0 & 0b1)    // 0
    fmt.Printf("%b\n", 0b1 & 0b0)    // 0
    fmt.Printf("%b\n", 0b1 & 0b1)    // 1

    // 只要左右有一個 bit 為 1 結果的 bit 就是 1
    fmt.Printf("%b\n", 0b0 | 0b0)    // 0
    fmt.Printf("%b\n", 0b0 | 0b1)    // 1
    fmt.Printf("%b\n", 0b1 | 0b0)    // 1
    fmt.Printf("%b\n", 0b1 | 0b1)    // 1

    // 左右 bit 只能有一個 1，不然結果的 bit 會是 0
    fmt.Printf("%b\n", 0b0 ^ 0b0)    // 0
    fmt.Printf("%b\n", 0b0 ^ 0b1)    // 1
    fmt.Printf("%b\n", 0b1 ^ 0b0)    // 1
    fmt.Printf("%b\n", 0b1 ^ 0b1)    // 0

    // 位元清除是一個比較特殊的運算
    // 他會先將右式取 not 再合起來
    fmt.Printf("%b\n", 0b0 &^ 0b0)    // 0 & 1 -> 0
    fmt.Printf("%b\n", 0b0 &^ 0b1)    // 0 & 0 -> 0
    fmt.Printf("%b\n", 0b1 &^ 0b0)    // 1 & 1 -> 1
    fmt.Printf("%b\n", 0b1 &^ 0b1)    // 1 & 0 -> 0
}
```

變數、資料型態、運算子

運算子

- 位元運算(and, or, xor, bit clear)
 - 這裡只拿一個位元舉例，但實際上在用時會搭配不只一個位元，可以自主練習一下：

```
package main
import "fmt"

func main(){
    fmt.Printf("%08b\n", 0b00001111 & 0b00110011) // 00000011
    fmt.Printf("%08b\n", 0b00001111 | 0b00110011) // 00111111
    fmt.Printf("%08b\n", 0b00001111 ^ 0b00110011) // 00111100
    fmt.Printf("%08b\n", 0b00001111 &^ 0b00110011) // 00001100
}
```


變數、資料型態、運算子

運算子

- 位元運算(左移、右移)
 - 左移、右移這個就有趣了，在十進位中，如果把數字左移一位，比如 15 左移一位變 150 左移兩位變 1500，也就是每左移一位數字會變 10 倍，反之每右移一位數字會變原先的十分之一。利用這個原理套用在 2 進位上，則數字每左移 1 位就會變 2 倍，左移兩位就會變 4 倍。

```
package main
import "fmt"

func main(){
    fmt.Printf("%04b\n", 0b0111 << 1)    // 1110
    fmt.Printf("%d\n", 15 << 1)           // 30

    fmt.Printf("%04b\n", 0b0111 >> 1)    // 0011
    fmt.Printf("%d\n", 15 >> 1)           // 7
}
```

變數、資料型態、運算子

輸入與輸出

- `fmt.Printf`：格式化輸出，用法跟C語言的`printf`一樣

```
a := 10
fmt.Printf("a: %d\n", a)
```

- 常用到的格式化輸出輸入參數

```
%d: digit    (10進位的數字)
%c: char     (字元)
%s: string   (字串)
%v: value    (值)
%+v 見下方
%#v 見下方
```

- 參數 `v` 三者的差異：<https://play.golang.org/p/UKM5CGI-AsR>
- 用於查看物件結構時，非常方便使用：

```
type Name struct {
    A string
    B bool
    C int
}

func main() {
    fmt.Printf("%v \n", Name{})
    fmt.Printf("%+v \n", Name{})
    fmt.Printf("%#v \n", Name{})
}

// { false 0}
// {A: B:false C:0}
// main.Name{A:"", B:false, C:0}
```

變數、資料型態、運算子

輸入與輸出

- fmt.Print、fmt.Println

- 兩者與fmt.Printf相比，差在不能印格式化輸出，Print與Println兩者主要差在ln多了一個換行(new line)

- 跳脫字元

" "雙引號 內可跳脫字元\t \n等

` `重音符 內則保留原始字串

```
func main() {  
    fmt.Print("\t \n")  
    fmt.Print(`\t \n`)  
}  
/* result:  
  
\t \n  
*/
```

```
func main() {  
    a := 10  
    fmt.Printf("a: %d\n", a)  
    fmt.Println("a: ", a)  
  
    s1 := "I"  
    s2 := "am"  
    s3 := "string"  
    fmt.Printf("%s%s %s\n", s1, s2, s3)  
    fmt.Println(s1 + s2 + s3)  
    fmt.Println(s1, s2, s3)  
  
    fmt.Println("=====")  
  
    fmt.Print(s1 + s2 + s3)  
    fmt.Print(s1, s2, s3)  
}  
/* result:  
a: 10  
a: 10  
Iam string  
Iamstring  
I am string  
=====  
IamstringIamstring  
*/
```

變數、資料型態、運算子

輸入與輸出

- Log日誌：log.Print、log.Println、log.Printf
 - Go的log.Print用法跟fmt.Print用法基本上一致，但log會在開一個線程出來，而fmt則是運行在主線程上：

```
func main() {  
    for i:= 0; i <= 3; i++){  
        log.Println(i)  
        fmt.Println(i)  
        // time.Sleep(time.Millisecond * 10)  
        log.Println("hi")  
        fmt.Println("hi")  
    }  
}
```

```
2020/09/13 18:05:16 1  
2020/09/13 18:05:16 hi  
2020/09/13 18:05:16 2  
2020/09/13 18:05:16 hi  
2020/09/13 18:05:16 3  
2020/09/13 18:05:16 hi  
1  
hi  
2  
hi  
3  
hi
```

```
2020/09/13 14:56:58 1  
1  
2020/09/13 14:56:58 hi  
hi  
2020/09/13 14:56:58 2  
2  
2020/09/13 14:56:58 hi  
hi  
2020/09/13 14:56:58 3  
3  
2020/09/13 14:56:58 hi  
hi
```

Process finished with exit code 0

- 開線程這點在Windows的環境上看的出來(左)；但在Playground及MacOS中運行起來是這樣(右)

變數、資料型態、運算子

輸入與輸出

- Log日誌：log.SetFlag
 - 改變印出的時間格式

```
log.SetFlags(0)
log.SetFlags(1)    2020/09/13
log.SetFlags(2)    20:57:00
log.SetFlags(3)    2020/09/13 20:57:00
log.SetFlags(4)    20:57:00.862816
log.SetFlags(5)    2020/09/13 20:57:00.862816
```

```
const (
    Ldate      = 1 << iota    // local time zone: 2009/01/23
    Ltime      //2           // local time zone: 01:23:23
    Lmicroseconds //4       // microsecond : 01:23:23.123123.
    Llongfile   //8         // full filename, line: /a/b/c/d.go:23
    Lshortfile   //16        // filename element, line: d.go:23
    LUTC         //32        // use UTC
    Lmsgprefix   //64        // move the "prefix" from the beginning of the line to before the message
    LstdFlags    = Ldate | Ltime //3 預設Flag // initial standard logger
)
```

- 可以用常數名稱直接帶入log.SetFlags(log.Ldate)，關於SetFlags的參數：
 - 若想要同時印出 檔案名稱 Lshortfile (16) 及 日期 Ldate(1)，則可直接相加數字，設定為log.SetFlags(17)，以此類推。
 - 也可以用| (或運算or) log.SetFlags(log.Lmsgprefix | log.LstdFlags)，格式可以自己兜，任君挑選！

```
import (
    "log"
)

func main() {
    for i := 0; i <= 127; i++ {
        log.SetFlags(i)
        log.Print("log.SetFlags(", i, ")")
    }
}
```

變數、資料型態、運算子

輸入與輸出

- Log日誌：log.SetPrefix

```
log.SetPrefix("文字")
```

```
func main() {  
    log.SetPrefix("安安，我是log ")  
    log.Println("Hi")  
    log.Fatalln("發生錯誤")  
}  
  
/* result:  
安安，我是log 2009/11/10 23:00:00 Hi  
安安，我是log 2009/11/10 23:00:00 發生錯誤  
*/
```

變數、資料型態、運算子

輸入與輸出

- Log日誌：log.Fatal、log.Fatalf、log.Fatalln
 - 用於發生錯誤時印出log並退出，點進去看go原始碼，退出的實作是多了os.Exit(1)這行

```
func Fatal(v ...interface{}) {  
    std.Output(2, fmt.Sprint(v...))  
    os.Exit(1)  
}
```

- 要自製logger，可以參考此處：<https://golang.org/pkg/log/#Logger>

變數、資料型態、運算子

輸入與輸出

- 如何讀取使用者的鍵盤(控制台)輸入呢？從鍵盤和標準輸入 `os.Stdin` 讀取輸入，最簡單的辦法是使用 `fmt` 套件提供的 `Scan` 和 `Sscan` 開頭的函數。請看以下程式：

```
// 從控制台讀取輸入：
package main
import "fmt"

var (
    firstName, lastName, s string
    i int
    f float32
    input = "56.12 / 5212 / Go"
    format = "%f / %d / %s"
)

func main() {
    fmt.Println("Please enter your full name: ")
    fmt.Scanln(&firstName, &lastName)
    // fmt.Scanf("%s %s", &firstName, &lastName)
    fmt.Printf("Hi %s %s!\n", firstName, lastName) // Hi Chris Naegels
    fmt.Sscanf(input, format, &f, &i, &s)
    fmt.Println("From the string we read: ", f, i, s)
    // 輸出結果: From the string we read: 56.12 5212 Go
}
```

Scanln：掃描來自標準輸入的資料，將空格分隔的值依次存放到後續的參數內，直到碰到換行。

Scanf：與其類似，除了 `Scanf` 的第一個參數用作格式字串，用來決定如何讀取。

Sscan 和以 **Sscan** 開頭的函數則是從字串讀取，除此之外，與 `Scanf` 相同。如果這些函數讀取到的結果與預想不同，可以檢查成功讀入資料的個數和返回的錯誤。