## Go features:

* Dynamic-typed language
* In-built GC.
* Concurrency programming: Go routines
* Case-sensitive language.

## Tutorials:

* 1. <https://www.tutorialspoint.com/go/index.htm>
  2. <https://golang.org/doc/tutorial/getting-started>
  3. <https://zalopay-oss.github.io/go-advanced/>
  4. <https://tour.golang.org/list>

## Syntax:

* Tokens: có thể là keyword, identifier, constant, string literal, symbol.
* Whitespace: blanks, tabs, new lines, comment. Dòng nào whitespace thì Go sẽ bỏ qua khi compile.
* Variable: mô tả vùng nhớ lưu trữ giá trị mà Go có thể tác động đến bằng thuật toán.

|  |
| --- |
| var i, j, k int;  var c, ch byte;  var f, salary float32;  d = 42; |

* Dynamic Type Declaration / Type Inference: compiler ko yêu cầu type tĩnh khi khai báo variable mà chỉ cần gán giá trị cho variable đó là đủ.
* Assignment operator: thuật toán gán, ko cần khai báo var. Nảy sinh khái niệm lvalue và rvalue: rvalue là giá trị được gán cho lvalue. Lvalue là variable, ko thể là constant.
* Khai bao const:

|  |
| --- |
| const <variable> <type> = <value>; |

Convention: định nghĩa constant bằng CAPITALS

* For loop: hỗ trợ 3 kiểu:

|  |
| --- |
| for [condition | ( init; condition; increment ) | Range] {  statement(s);  } |

* + condition: giống while
  + init; condition; increment: giống C
  + range:

|  |
| --- |
| numbers := [6]int{1, 2, 3, 5}  for i,x:= range numbers {  fmt.Printf("value of x = %d at %d\n", x,i)  } |

* Functions:
  + Hỗ trợ return multiple results:

|  |
| --- |
| package main  import "fmt"  func swap(x, y string) (string, string) {  return y, x  }  func main() {  a, b := swap("Mahesh", "Kumar")  fmt.Println(a, b)  } |

* + Hỗ trợ 2 hình thức pass parameter:
    - value: default.
    - reference: giống C, truyền pointer variable vào parameter của function. Thay vì truyền value vào function parameter thì truyền pointer vào.
  + Khai báo và sử dụng function:
    - As value:

|  |
| --- |
| package main  import ("fmt" "math")  func main(){  /\* declare a function variable \*/  getSquareRoot := func(x float64) float64 {  return math.Sqrt(x)  }  /\* use the function \*/  fmt.Println(getSquareRoot(9))  } |

* + - Closure:

|  |
| --- |
| package main  import "fmt"  func getSequence() func() int {  i:=0  return func() int {  i+=1  return i  }  }  func main(){  /\* nextNumber is now a function with i as 0 \*/  nextNumber := getSequence()  /\* invoke nextNumber to increase i by 1 and return the same \*/  fmt.Println(nextNumber())  } |

* + - Method: dùng receiver để làm container của method. Ở đây receiver = circle.

|  |
| --- |
| package main  import (  "fmt"  "math"  )  /\* define a circle \*/  type Circle struct {  x,y,radius float64  }  /\* define a method for circle \*/  func(circle Circle) area() float64 {  return math.Pi \* circle.radius \* circle.radius  }  func main(){  circle := Circle{x:0, y:0, radius:5}  fmt.Printf("Circle area: %f", circle.area())  } |

* + Function có hỗ trợ parameter dạng linh động số lượng (...), kiểu parameter này sẽ là Slice.

|  |
| --- |
| // Biến số lượng tham số 'more'  // Tương ứng với kiểu [] int, là một slice  func Sum(a int, more ...int) int {  for \_, v := range more {  a += v  }  return a  } |

* String: Khá giống Java, string là immutable, khởi tạo 1 lần và ko thể thay đổi.

Kiểu: slice.

* Pointer: pointer là 1 biến mà lưu trữ địa chỉ của 1 biến khác, ví dụ là địa chỉ trực tiếp của vị trí bộ nhớ.

Hỗ trợ pointer variable trỏ đến 1 pointer variable khác, giống C.

* Slice: là trừu tượng hóa của array trong Go, cung cấp khả năng tăng size và 1 số thao tác với array.
* Type conversion: kieu type la function, tham so la gia tri can convert.

|  |
| --- |
| <type\_name>(<expression>) |

* Interface: 1 data type riêng, dùng để biểu diễn các method signature. Struct nào implement interface nào sẽ implement các method này. Các bước implement interface:
  + Định nghĩa interface.
  + Định nghĩa struct nào sẽ implement interface này.
  + Định nghĩa các method implementation, parameter của method này là struct variable, tên method trùng với tên method khai báo ở interface.

|  |
| --- |
| /\* define an interface \*/  type <interface\_name> interface {  method\_name1 [return\_type]  method\_name2 [return\_type]  method\_name3 [return\_type]  ...  method\_namen [return\_type]  }  /\* define a struct \*/  type <struct\_name> struct {  /\* variables \*/  }  /\* implement interface methods\*/  func (<struct\_name\_variable> <struct\_name>) method\_name1() [return\_type] {  /\* method implementation \*/  }  ...  func (<struct\_name\_variable> <struct\_name>) method\_namen() [return\_type] {  /\* method implementation \*/  } |

* Error Handling: su dung inbuilt interface type.

|  |
| --- |
| package main  import "errors"  import "fmt"  import "math"  func Sqrt(value float64)(float64, error) {  if(value < 0){  return 0, errors.New("Math: negative number passed to Sqrt")  }  return math.Sqrt(value), nil  }  func main() {  result, err:= Sqrt(-1)  if err != nil {  fmt.Println(err)  } else {  fmt.Println(result)  }    result, err = Sqrt(9)  if err != nil {  fmt.Println(err)  } else {  fmt.Println(result)  }  } |

* Defer trong Function: hoãn việc thực thi hàm cho tới khi hàm bao ngoài nó return. Mỗi lời gọi defer được push vào stack và thực thi theo thứ tự ngược lại khi hàm bao ngoài nó kết thúc. Ta thường sử dụng defer cho việc đóng hoặc giải phóng tài nguyên:

|  |
| --- |
| func main() {  defer fmt.Println("world")  fmt.Println("hello")  }  // kết quả: hello world |

* + Đóng file giống như try-finally:

|  |
| --- |
| func main() {  f, err := os.Create("file")  if err != nil {  panic("cannot create file")  }  // chắc chắn file sẽ được close dù hàm có bị panic hay return  defer f.Close()  fmt.Fprintf(f,"hello")  } |

* + Đóng file và xử lý panic giống như try-catch-finally:

|  |
| --- |
| func main() {  defer func() {  msg := recover()  fmt.Println(msg)  }()  // . là folder hiện tại  f, err := os.Create(".")  if err != nil {  panic("cannot create file")  }  defer f.Close()  // không quan trọng chuyện gì xảy ra thì file cũng sẽ được close  // để đơn giản nên ở đây bỏ qua bước kiểm ra close result  fmt.Fprintf(f,"hello")  } |

* + Cũng giống như block finally thì lời gọi defer cũng có thể làm cho kết quả trả về thay đổi:

|  |
| --- |
| func yes() (text string) {  defer func() {  text = "no"  }()  return "yes"  }  func main() {  fmt.Println(yes())  } |

* a

## Data Types:

* Integer: uint8, uint16, uint32, uint64, int8, int16, int32, int64

int8: Signed 8-bit integers (-128 to 127).

* Float: float32, float64, complex64, complex128
* Others: byte = uint8, rune = int32, uint, int, uintptr.

## Project Layout:

* 1. Ref: <https://github.com/golang-standards/project-layout>

Có thể coi như đây là các package chính của application Go.

* 1. Go Directories:
     1. /cmd: Main applications for this project. The directory name for each application should match the name of the executable you want to have (e.g., /cmd/myapp).
     2. /internal: Private application and library code. This is the code you don't want others importing in their applications or libraries. Note that this layout pattern is enforced by the Go compiler itself.
     3. /pkg: Library code that's ok to use by external applications (e.g., /pkg/mypubliclib). The /pkg directory is still a good way to explicitly communicate that the code in that directory is safe for use by others.
     4. /vendor: Application dependencies.
  2. Service Application Directories:
     1. /api: OpenAPI/Swagger specs, JSON schema files …
  3. Web Application Directories:
     1. /web: static web assets, server side templates and SPAs.
  4. Common Application Directories:
     1. /configs:
     2. /init: System init and process manager/supervisor configs.
     3. /scripts: Scripts to perform various build, install, analysis …
     4. /build: Packaging and Continuous Integration. Put your cloud (AMI), container (Docker), OS (deb, rpm, pkg) package configurations and scripts in the /build/package directory.
     5. /deployments:
     6. /test:
  5. Directories You Shouldn't Have:
     1. /src:

## Training on Examples:

* 1. Dựng REST Api đơn giản:

<https://viblo.asia/p/tim-hieu-ve-golang-qua-example-restful-api-su-dung-gorilla-mux-RnB5pbNGZPG>

* 1. Tích hợp Postgresql:

<https://semaphoreci.com/community/tutorials/building-and-testing-a-rest-api-in-go-with-gorilla-mux-and-postgresql>

* 1. RESTful Web API CRUD using GORM in Golang:

<https://viblo.asia/p/restful-web-api-crud-using-gorm-in-golang-4P856nRA5Y3>

* 1. GORM: <https://viblo.asia/p/restful-web-api-crud-using-gorm-in-golang-4P856nRA5Y3>
  2. Tạo RESTful API với Golang và MongoDB: <https://viblo.asia/p/tao-restful-api-voi-golang-va-mongodb-ORNZq44GK0n>
  3. Create Your First Rest API With GOLANG using GIN, GORM and MySql : <https://medium.com/wesionary-team/create-your-first-rest-api-with-golang-using-gin-gorm-and-mysql-d439bcc6f987>

<https://github.com/SudeepTimalsina/first-go>

* 1. Gin awesome: <https://awesomeopensource.com/project/FlowerWrong/awesome-gin>
  2. a

## Thư viện:

* 1. Postgresql Connection: <https://github.com/go-pg/pg>
  2. GORM: <https://github.com/go-gorm/gorm>
  3. Kafka-go: <https://github.com/segmentio/kafka-go>
  4. Gin-swagger: <https://github.com/swaggo/gin-swagger>
  5. Zerolog: <https://github.com/rs/zerolog>
  6. Goconvey: <https://github.com/smartystreets/goconvey/>
  7. Testify: <https://github.com/stretchr/testify>
  8. RabbitMQ: <https://github.com/streadway/amqp>
  9. a

## a