API WITH GO

Golang programing

Building a Basic REST API in Go using Fiber

Golang

Go is programming language designed and supported by Google by Robert Griesemer, Rob Pike and Ken Thompson. First appeared November 10, 2009

Go: installed

https://go.dev/doc/install

Golang:
Basic

Go: Say “Hello World”

This should be the first line of code. So “main” is name of the package which this file belong to

```go
package main

import "fmt"

func main() {
    fmt.Println("\"Hello\"")
}
```

“func main” is a special func tells Golang here to start executing and only stand on package main

If package isn’t main, you will see: package command-line-arguments is not a main package

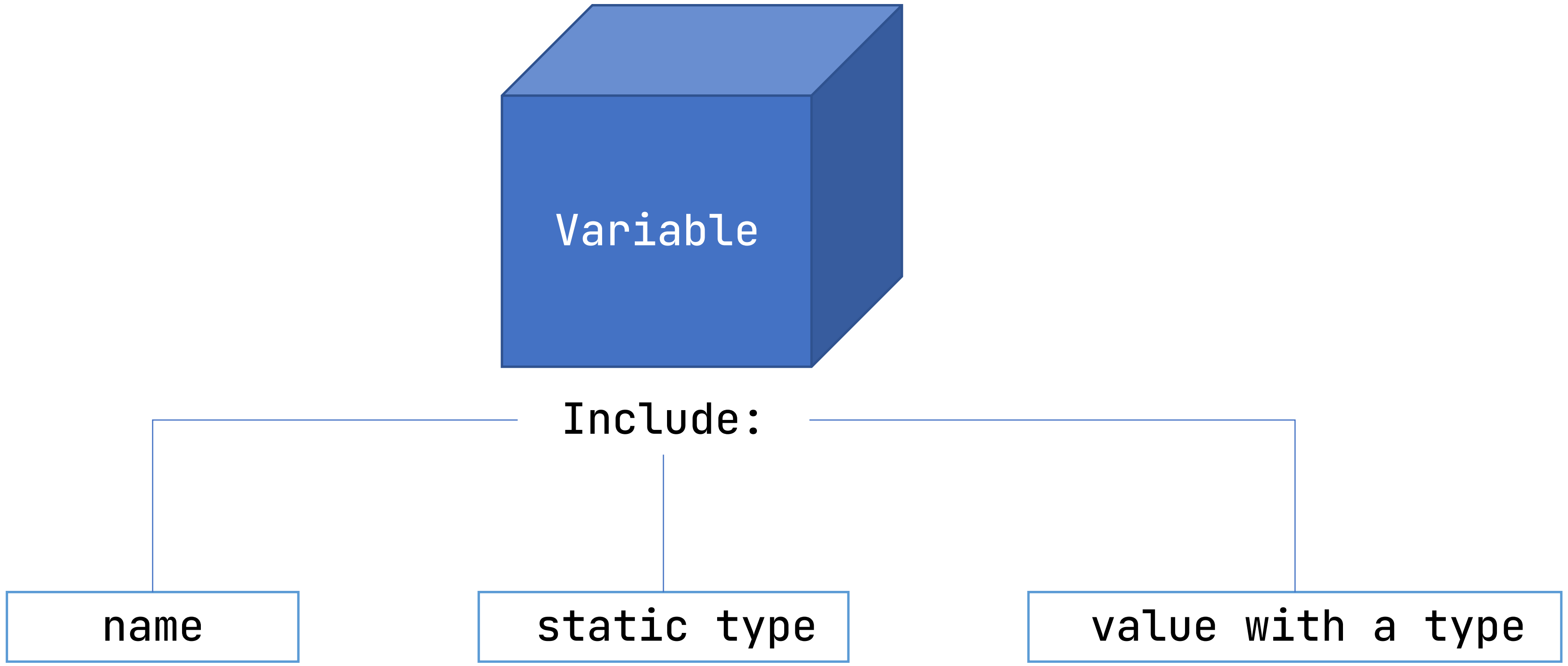
Go: Data Type

Type	Description
Boolean	consists of the two predefined constants: “true” and “false”
Numeric	Represents: integer types or floating point values throughout the program.
String	represents a string value
Derived	Include: Pointer types, Array types, Structure types, Union types and Function types, Slice types, Interface types, Map types, Channel Types

Go: Basic Data Type

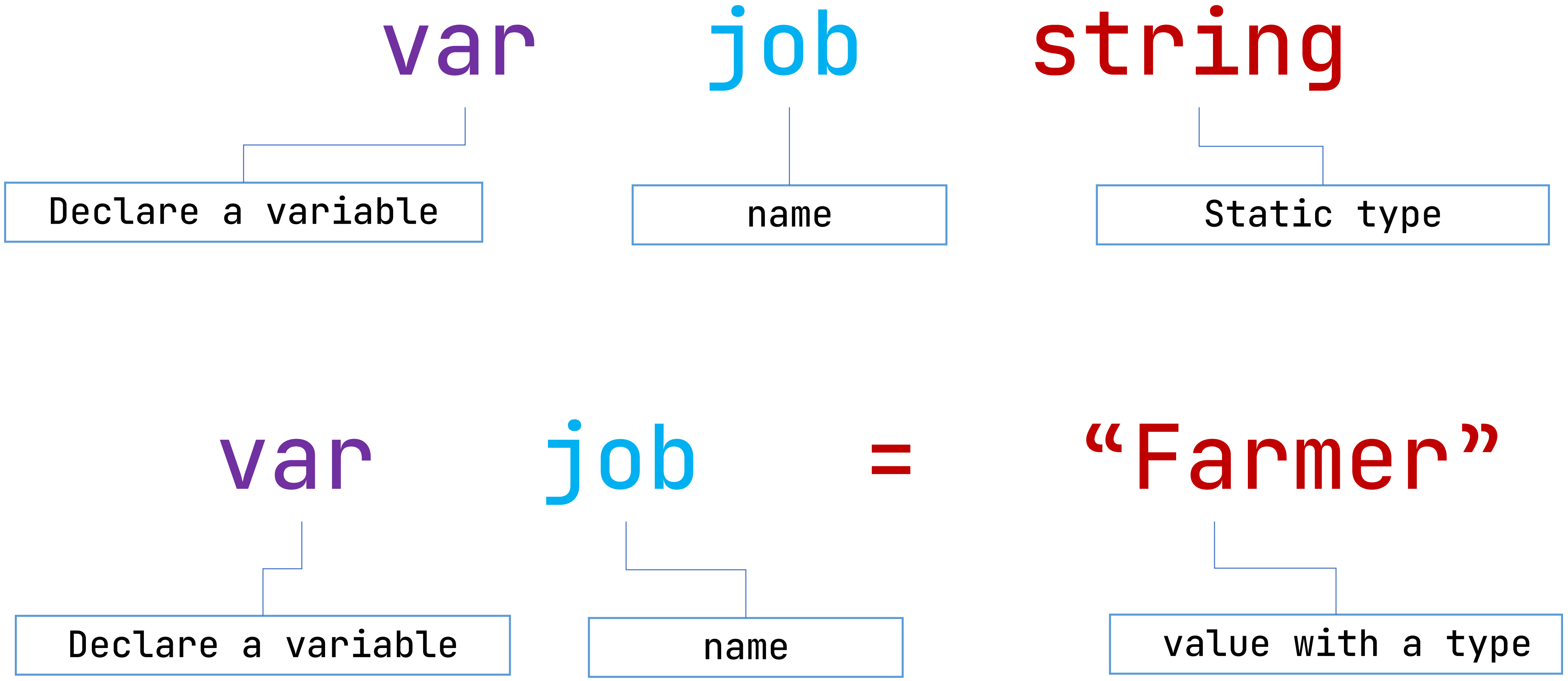
Type	Value literals
int	67, -1, 0, 33
float64	-0.5, 0.0, 1.0, 20.40
bool	true, false
string	“Hello World”, “call 1150”

Go: Declaring Variables



Go: Declaring Variables

var declaration

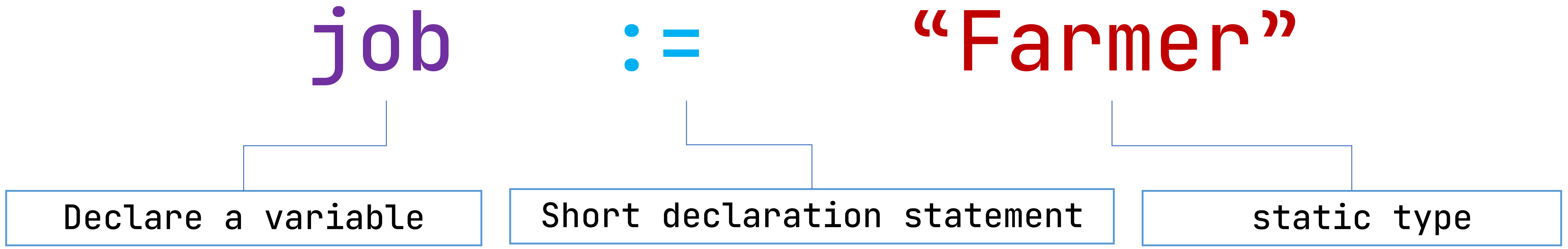


```go
var job string
```

```go
var job = “Farmer”
```

Go: Declaring Variables

short declaration



```
job := “Farmer”
```

We can’t use short declaration at the package scope.

Go: Declaring Variables

Unused variable

We can declared and not used on package-scope.

```go
package main

import "fmt"

var project string

func main() {
    var database string
    _ = database
}
```

We can't declared and not used on function-scope.
Admit the values by using the blank-identifier.

Go: Declaring Variables

etc.

```go
package main

import "fmt"

// var [variable_name] [type] = [value] or var [variable_name] =
[type]
var project string

var (
    dbhost   string = "127.0.0.1"
    dbport          = "5432"
    dbname   string
    dbuser   string
    dbsecret string
)

func main() {
    // [variable_name] := [value]
    domain := "127.0.0.1"
    port := "3333"

    fmt.Printf("Base URL: %s:%s", domain, port)
}
```

Note: If a variable should have a fixed value that cannot be changed, you can use the **const** keyword.

Go: if statement

```go
package main

import "fmt"

func main() {
    var score = 87
    if score >= 90 {
        fmt.Println("A")
    } else if score >= 75 {
        fmt.Println("B")
    } else if score >= 60 {
        fmt.Println("C")
    } else if score >= 50 {
       fmt.Println(“D")
    } else {
        fmt.Println(“F")
    }
}
```

if statement in Golang
that doesn’t require parentheses.

If statement’s block is executed only if its condition expression is “true”

Statement in “If statement’s block” are only visible inside the “if block” (curly bracket)

Go: if statement

```go
package main

import "fmt"

func main() {
    var score = 87
    if score >= 90 {
        fmt.Println("A")
    } else if score >= 75 {
        fmt.Println("B")
    } else if score >= 60 {
        fmt.Println("C")
    } else if score >= 50 {
       fmt.Println(“D")
    } else {
        fmt.Println(“F")
    }
}
```

“if else” will executed if previous branches are false

“else” will executed if all the branches are false

Go: Array in Go

This will happened the error

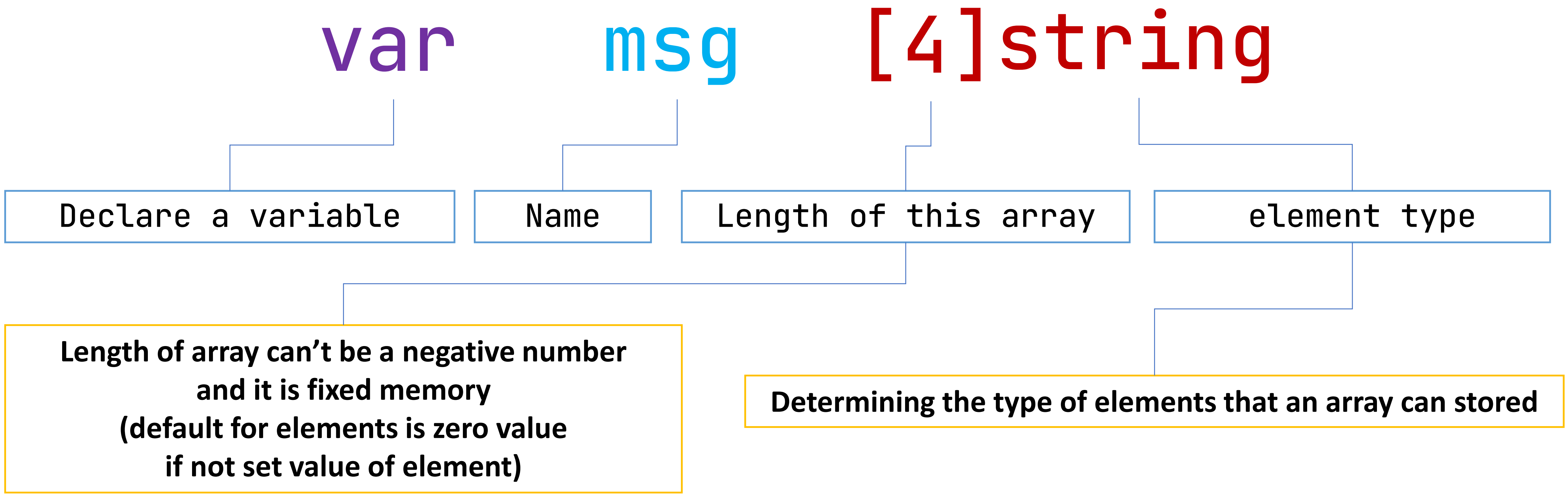
```go
package main

import "fmt"

func main() {
    var msg []string
    msg[0] = "Hello"
    fmt.Println(msg)
}
```

Pass

```go
package main

import "fmt"

func main() {
    var msg [1]string
    msg[0] = "Hello"
    fmt.Println(msg)
}
```

Array in Go
It is a fix length for stored
And element on array as only the same type of values
(an array stores its elements in contiguous memory cells)

Go: Array in Go



```go
var msg [4]string
```

Length of array can’t be a negative number
and it is fixed memory
(default for elements is zero value
if not set value of element)

Determining the type of elements that an array can stored

Go: Array in Go

Get and Set the element of array

```go
package main

import "fmt"

func main() {
    var msg = [4]string{
        "Hello",
        "Hey",
    }
    msg[2] = "Goodbye"
    fmt.Println(msg[1] + ", " + msg[2])
}
```

An Array has key of elements called “index” that start at 0

We can get an array element with an index expression

```go
msg[1] // "Goodbye"
```

We can set an array element with an index expression and assignment operators

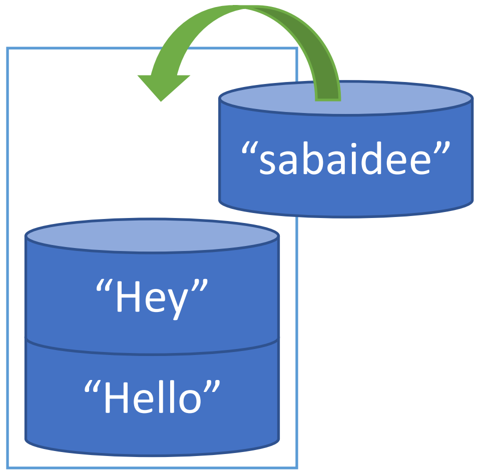
```go
msg[2] = "Goodbye"
```

Go: Slice

Differences between slices and arrays

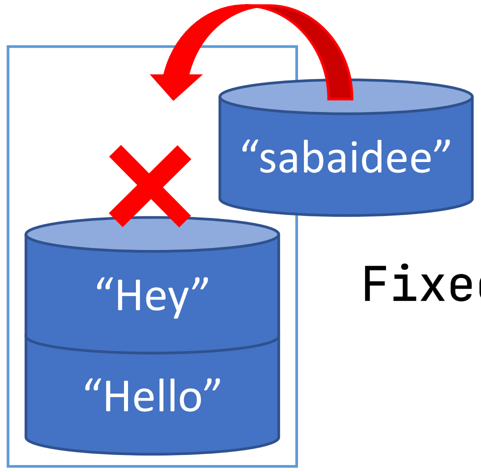
```
var msg [] string
```

Slice can grow and shrink in runtime
And doesn’t fixed length at runtime



```
var msg [2]string
```

Array can’t grow and shrink in runtime
And fixed length at runtime



Fixed length

Go: Slice

Get and Set element of slice

```go
package main

import "fmt"

func main() {
    var msg = []string{
        "Hello",
        "Hey",
    }
    msg = append(msg, "Goodbye")
    fmt.Println(msg)
    fmt.Println(msg[2])
}
```

We can't get and set non-existing elements in slice such as
"`fmt.Println(msg[4])`"
This will happened the error.

Like as array, For slice can only the same type of elements.

We can new element to a slice with "append", "append" can't change the passed slice but it return a new slice.

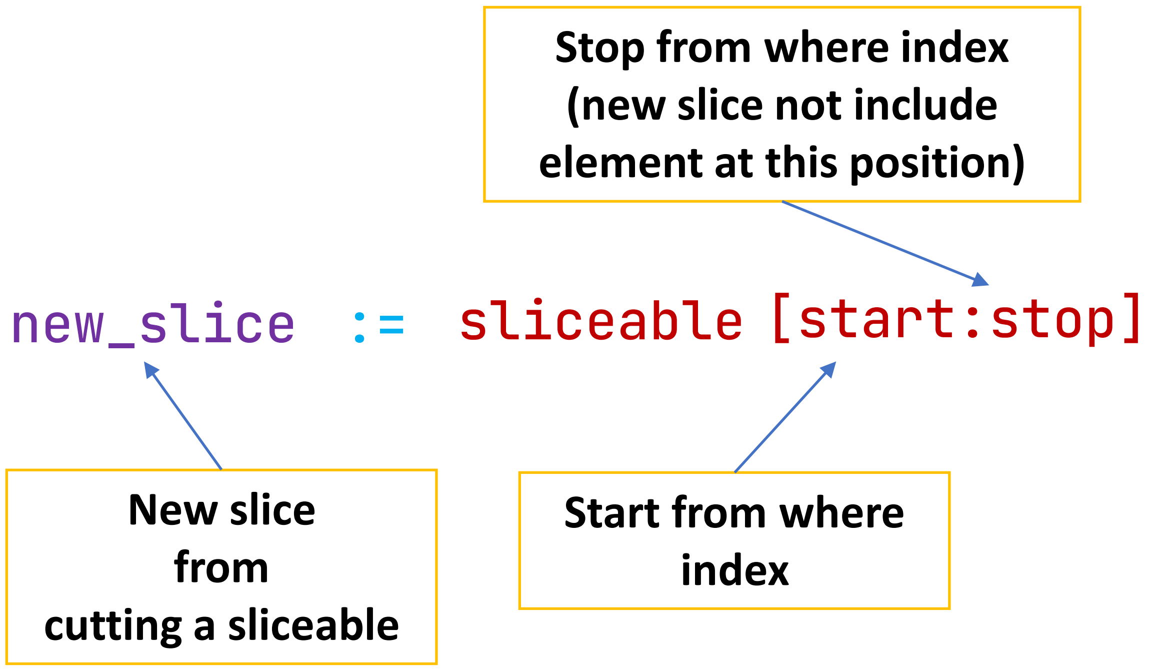
We can get element with index expression like as array.

Go: Slice

Slice expression

```go
package main

import "fmt"

func main() {
    var msg = []string{
        "Hello",
        "Hey",
    }
    msg = append(msg, "Goodbye")
    msg = msg[1:3] // Or msg = msg[1:]
    fmt.Println(msg)
}
```

Slice creates a new slice by cutting a sliceable



Stop from where index (new slice not include element at this position)

`new_slice := sliceable [start:stop]`

New slice from cutting a sliceable

Start from where index

Go: For Loop

```go
package main

import "fmt"

func main() {
	fmt.Println(SumBetweenNumber(9, 12))
}

func SumBetweenNumber(start int, end int) int {
	sum := 0
	for i := start; i <= end; i++ {
		sum += i
	}
	return sum
}
```

“For statement” will repeat a statement inside the block as long as its condition is true.

“;” is separator the parts of a “for statement”.

“i := start” (init statement) will be executed before the for loop begins.

“i <= end” (condition expression) will be checked just before each loop step start.

“i ++” (post statement) will be executed after each step of the loop.

Go: For Loop

```go
package main

import "fmt"

func main() {
    fmt.Println(SumBetweenNumber(9, 12))
}

func SumBetweenNumber(start int, end int)
int {
    sum := 0
    for ; start <= end; start++ {
        sum += start
    }
    return sum
}
```

"for statement" can non-existing "init statement" or "post statement"

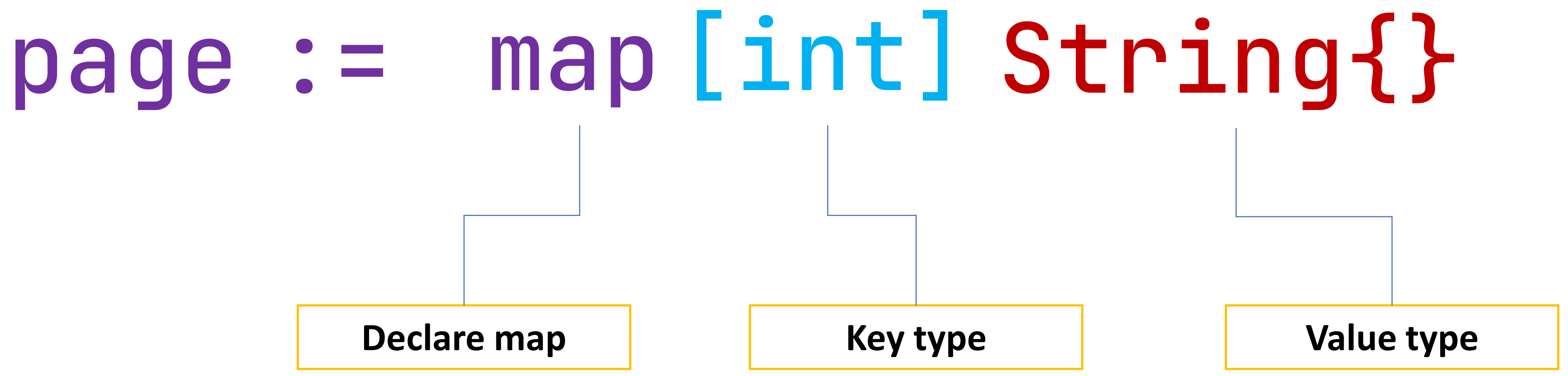
Go: For Loop

```go
package main

import "fmt"

func main() {
    fmt.Println(SumBetweenNumber(9, 12))
}

func SumBetweenNumber(start int, end int)
int {
    sum := 0
    for {
        if start > end {
            break
        }
        sum += start
        start++
    }
    return sum
}
```

We can exist froom the loop by using the break statement

Go: Maps

Maps can access to an element with a unique key (Map key must be unique).



```go
page :=  map[int]String{}
```

Go: Maps

Key must be a comparable type

```
map [[]string]string
```

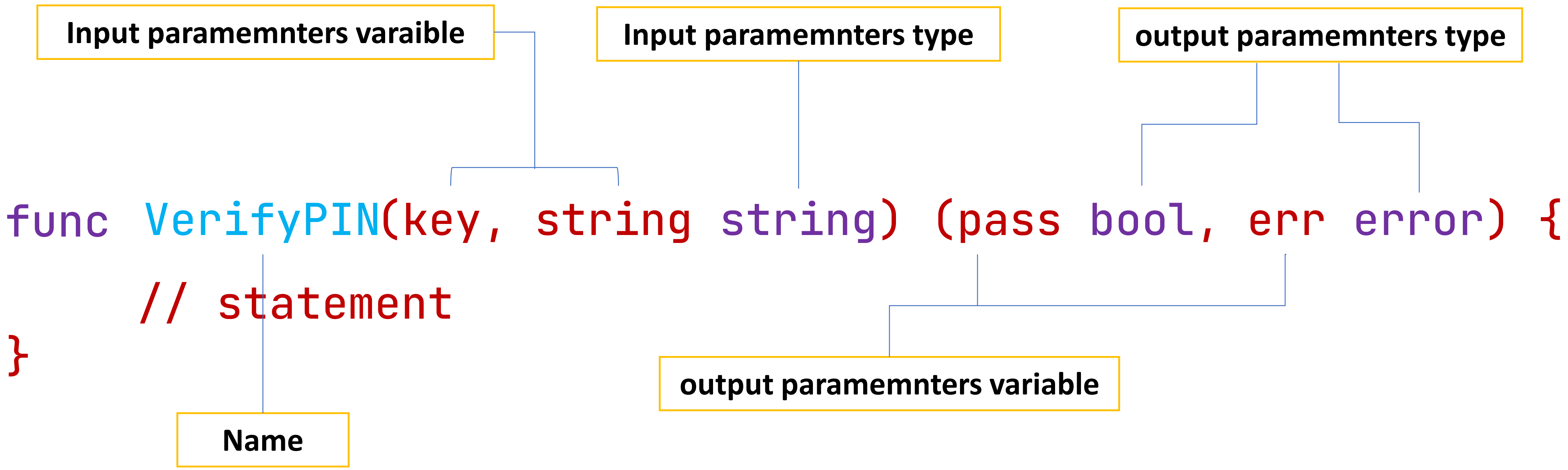
Map is incorrect,
Slice, map, and function are not comparable.

Go: Maps

```go
package main

import "fmt"

func main() {
    var greeting map[string]string = map[string]string{"fr": "Bonjour"}
    greeting["en"] = "Hello"
    greeting["es"] = "Hola"
    greeting["de"] = "Hallo"
    fmt.Println(greeting["en"])
    fmt.Println(greeting["de"])
    fmt.Println(greeting["fr"])
}
```

Maps can change if it’s been initialized if the above code as
“`var greeting map[string]string`”
Will happened error.

Go: function



```go
func VerifyPIN(key, string string) (pass bool, err error) {
     // statement
}
```

Go: function

```go
package main

import (
    "fmt"
    "strings"
)

func main() {
    msg := Greeting("Kham", "seangphachanh")
    fmt.Println(msg)
}

func Greeting(name string, surname string) string {
    return fmt.Sprintf("Hello, %s %s", name, strings.ToUpper(surname))
}
```

"func" is the keyword for declaring a function.
Every package-level function has a name ("init" and "main" name is reserved)
Function might has input parameters, output parameters or non-existing,
Statement in function can only be visible inside self.

Go: Pointer

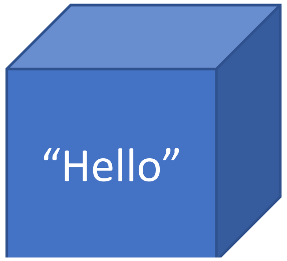
Pointer has stored the memory address of a value.

p is pointer variable and v is variable.

```
var  greeting string = “Hello”
```

```
p := &greeting
```

“p” store address of “Hello”
(xc0909dx)
(A pointer to a string value)



Address: xc0909dx

```
v:= *p
```

v copy value from the position at "p" direct to greeting variable

Go: Pointer

```go
package main

import "fmt"

func main() {
    var greeting string = "Hello"
    p := &greeting
    fmt.Println("p variable value: ", *p)
    fmt.Println("greeting variable value: ", greeting)
    *p = "Bye"
    fmt.Println("p variable value: ", *p)
    fmt.Println("greeting variable value: ", greeting)
}
```

`*p = "Bye“`

p is a string pointer variable (*string) that point t a string variable (greeting)

So now value of greeting equal “Bye” and

`p := &greeting`

because p store memory address of greeting variable, now *p equal “Bye” like as a greeting variable.

Go: Pointer

```go
package main

import "fmt"

type Province struct {
    Name string `json:"name"`
}

func main() {
    p := Province{Name: "Vientiane"}
    NewProvinceName(&p.Name, "VTC")
    fmt.Println(p)
}
func NewProvinceName(old *string, new string)
{
    *old = new
}
```

```go
package main

import "fmt"

type Province struct {
    Name string `json:"name"`
}

func main() {
    p := Province{Name: "Vientiane"}
    NewProvinceName(p.Name, "VTC")
    fmt.Println(p)
}
func NewProvinceName(old string, new string) {
    old = new
}
```

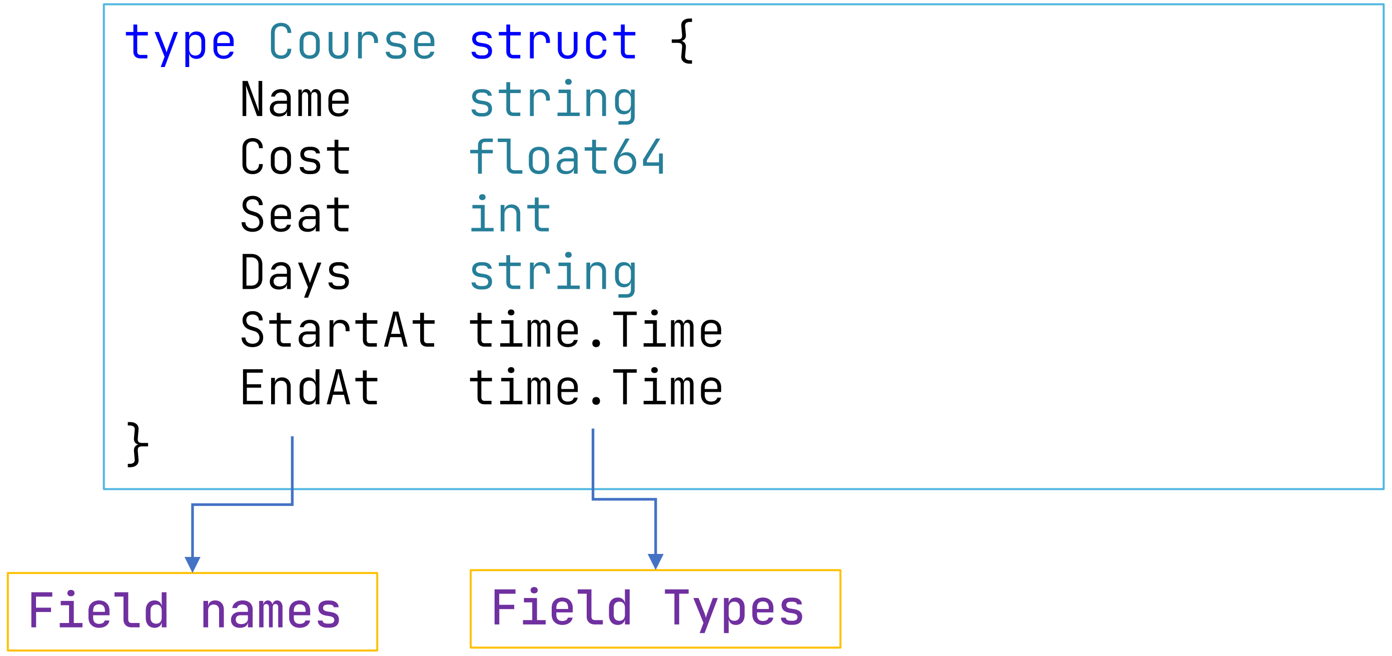
Go: Struct

Filed Names	Field Types	Field Values
Name	string	“Nodejs”
Cost	float64	“780000.00”
Seat	int	“20”
Duration	string	“"Mon, Tue, Wed, Thu, Fri"
StartAt	time.Time	“2006-01-02 15:04:05”
EndAt	Time.Time	“2006-01-05 15:04:05”

- **Struct is a collection of field.**
- **Struct is blueprint.**
- **It’s like a class in OOP Language.**
- **Group related attribute.**
- **Fixed at complie-time.**

Go: Struct

Struuct can’t dynamically grow but they can have difference set of type so not like slice and map.
A struct may store different types of data.



```
type Course struct {
    Name    string
    Cost    float64
    Seat    int
    Days    string
    StartAt time.Time
    EndAt   time.Time
}
```

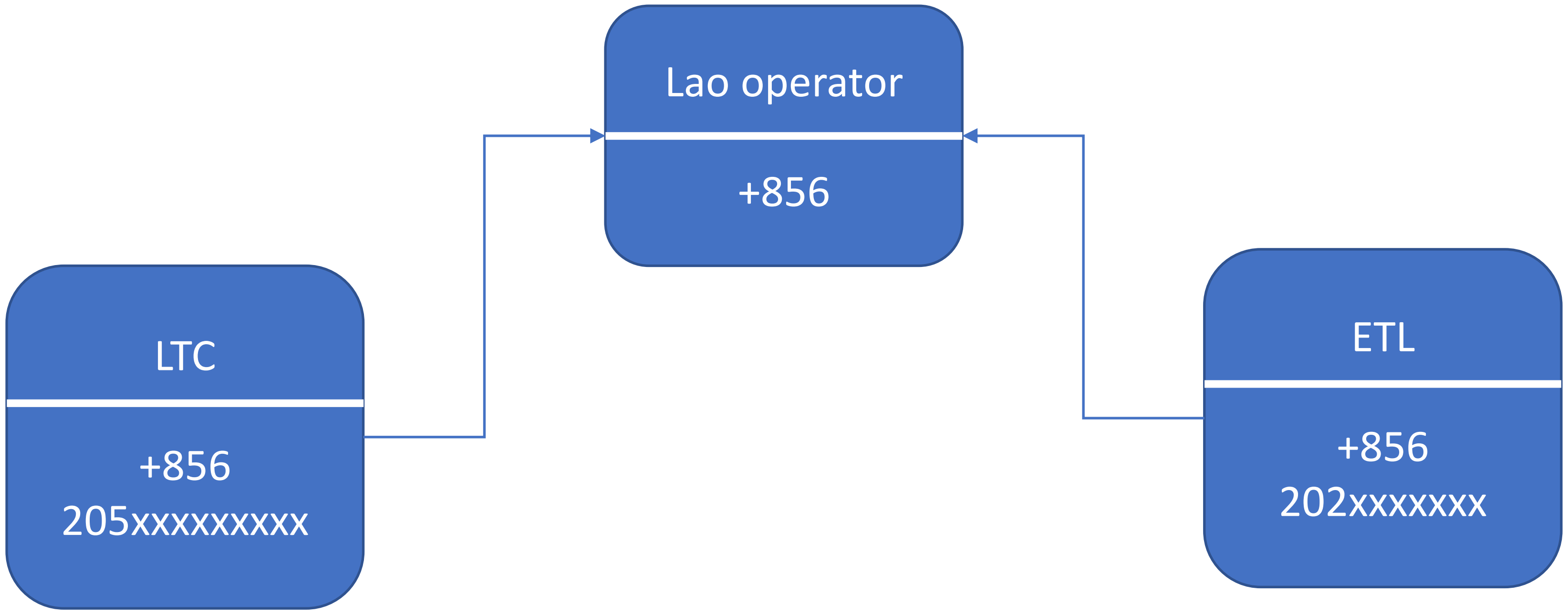
Go: Struct

```go
package main
import (
    "fmt"
    "time"
)
type Course struct {
    Name    string
    Cost    float64
    Seat    int
    Days    string
    StartAt time.Time
    EndAt   time.Time
}
func main() {
    course := Course{
        Name: "Go", Cost: 100, Seat: 10, Days: "Mon, Tue, Wed, Thu, Fri",
        StartAt: time.Now(), EndAt: time.Date(2022, 11, 17, 20, 34, 58, 0, time.UTC),
    }
    fmt.Println(course)
}
```

Go: Struct

Inheritance

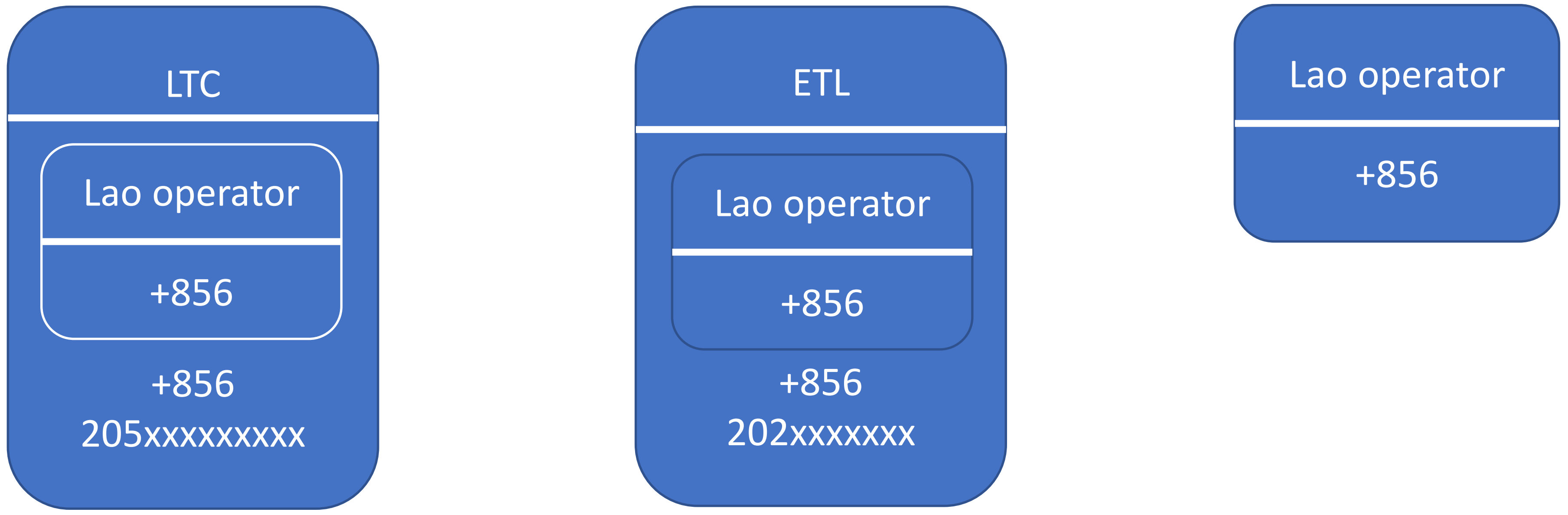
“is-a” relations:
LTC is Lao phone operator and Dim-sums is a Lao phone operator.



Go: Struct

Embedding

“has-a” relations:
LTC has a lao operator and ETL has a lao operator



Go: Struct

Field tag

Field tag is associating a static string metadata to a field.
Mostly used for controlling the encoding/decoding behavior.

```
type Course struct {
    Name    string          `json:”name” firestore:”name”`
    Cost    float64         `json:”cost” firestore:”cost”
}
```

Go: Method

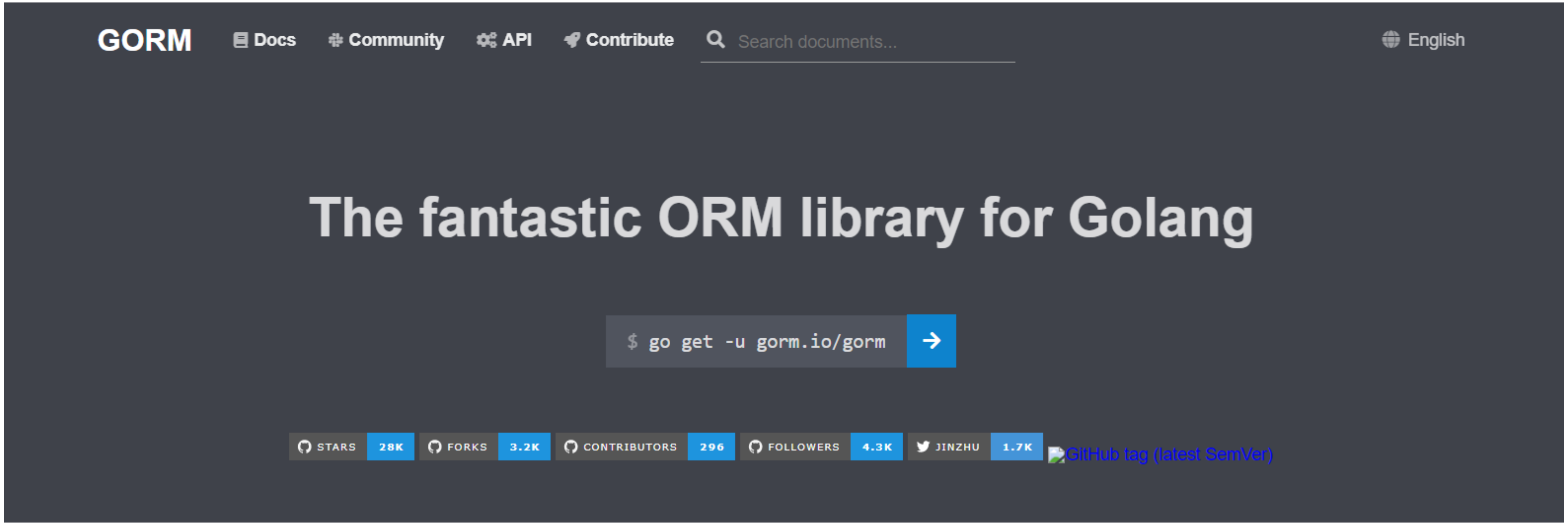
```go
package main

import "fmt"

type msg string

func (m *msg) Display() {
    fmt.Println(*m)
}

func main() {
    m := msg("Hello, World!")
    m.Display()
}
```

We can define methods on types.
A method is a function with a special *receiver* argument.
***Remember:* a method is just a function with a receiver argument.**

ORM

is a technique that lets you query and manipulate data from a database using an object-oriented paradigm. converting data between type systems using object-oriented programming languages.

GORM



The fantastic ORM library for Golang

```
$ go get -u gorm.io/gorm
```

Connecting with the database

Gorm is helper functions to communicate with the database. we will using Postgres database but GORM also supports MySQL, SQLite, and other SQL databases (see document more).

```go
// Example:
func NewDB() (*gorm.DB, error) {
    db, err := gorm.Open(postgres.Open(DSN), &gorm.Config{})
    if err != nil {
        return nil, fmt.Errorf("Failed to connect to database: %v", err)
    }
    return db, nil
}
```

Gorm: Model

This creates, in effect, a "virtual object database" that can be used from within the programming language (see document more).

```go
// Example:
type DistrictDB struct {
    gorm.Model
    Name       string `gorm:"type:text;not null"`
    NameEn     string `gorm:"type:text"`
    ProvinceID uint
    Province   ProvinceDB `gorm:"constraint:OnDelete:CASCADE;"`
}
```

Gorm: CRUD

Which stands for four functions: Create/Read/Update/Delete. It allows to create an object and save it in a database, to get an objects from a database, and to update and delete an object (see document more).

```
// Example:

“DB.Create(&p)”

“DB.Find(&provinces)”

“DB.Model(&province).Where("id = ?", p.ID).Updates(&p)”

“DB.Where("id = ?", id).Delete(&province)”
```

Gorm: Migration

Migrate your database with gorm (also database migration, database change management). we can performed on a database whenever it is necessary to update or revert that database's schema.(see document more).

```go
// Example:
func MigrateDB(db *gorm.DB) error {
    var err error
    err = db.AutoMigrate(&repository.ProvinceDB{}, &repository.DistrictDB{}, &repository.VillageDB{})
    if err != nil {
        return fmt.Errorf("Failed to migrate database: %v", err)
    }
    // Drop foreign key for rename.
    err = db.Migrator().DropConstraint(&repository.VillageDB{}, "fk_village_dbs_district")
    if err != nil {
        return fmt.Errorf("Failed to drop constraint: %v", err)
    }
    // New foreign key.
    err = db.Migrator().CreateConstraint(&repository.VillageDB{}, "fk_village_dbs_district")
    if err != nil {
        return fmt.Errorf("Failed to add constraint: %v", err)
    }
    return nil
}
```