GOLANG BASIC

2024, 23 February

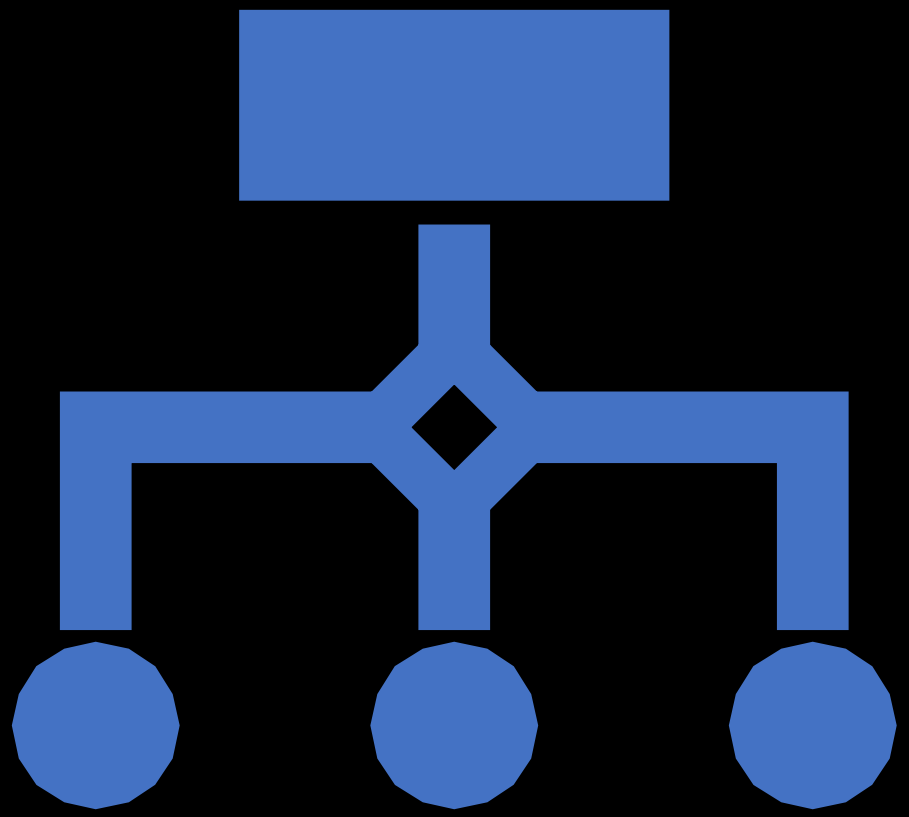
Outline

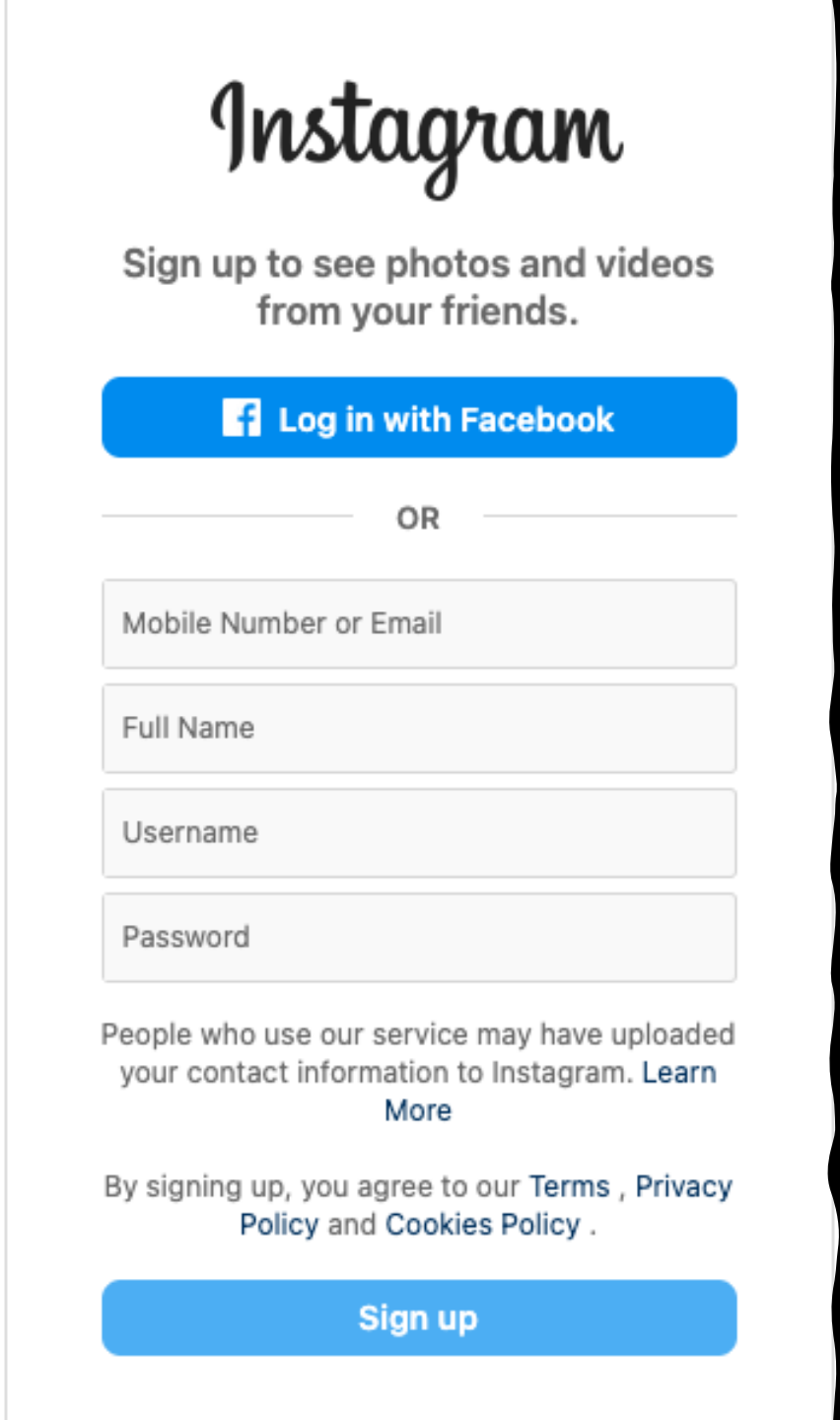
- Install
- Declarations
  - Variable
  - Constanta
  - Data Types
- Input & Output
  - Print Output
  - Take Input
- Conversion & Assertion
  - Type Conversion
  - Type Assertion
- Comments
- Flow Control
  - Operators
  - Conditional Statements
  - Loop Statements
- Collection Types
  - Array
  - Slice
  - Map
- Empty Type
- Function
  - Parameter
  - Return Value
  - Return Predefined Value
  - Variadic
- Access Modifier
  - Public Property
  - Private Property
- Declaration Types
  - Struct
  - Interface
- Method
- Pointer
- Defer, Error, Panic, Recover

FLOW CONTROL

Flow Control

- The control flow is the order in which the computer executes statements in a script.
- Code is run in order from the first line in the file to the last line, unless the computer runs across the structures that change the control flow, such as conditionals and loops.





Example Flow Control

- Imagine a script used to validate user data from a webpage form.
- The script submits validated data, but if the user, say, leaves a required field empty, the script prompts them to fill it in.
- To do this, the script uses a conditional structure or if … else, so that different code executes depending on whether the form is complete or not.

Flow Control

OPERATORS

CONDITIONAL STATEMENTS

LOOP STATEMENTS

Operators

- Arithmetic Operators
- Comparison Operators
- Logical Operators

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Arithmetic Operators

```go
func main() {
  a, b := 4, 2
  x := (a + b) / (a - b) * 2
  fmt.Println(x)

  y := 9 % a
  fmt.Println(y)
}
```

Arithmetic Operators

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/flow_control/operator/arithmetic.go"
6
1

[Done] exited with code=0 in 0.514 seconds
```

Augmented Assignments

Operator	Augmented Assignments
a = a + 10	a += 10
a = a - 10	a -= 10
a = a * 10	a *= 10
a = a / 10	a /= 10
a = a % 10	a %= 10

Augmented Assignments

```go
func main() {
  a, b := 2, 3

  // increment assignment
  a += b
  fmt.Println(a)

  // decrement assignment
  b -= 2
  fmt.Println(b)

  // multiplication assignment
  b *= 10
  fmt.Println(b)

  // division assignment
  b /= 5
  fmt.Println(b)

  // modules assignment
  a %= 3
  fmt.Println(a)
}
```

Augmented Assignments

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/flow_control/operator/augmented.go"
5
1
10
2
2

[Done] exited with code=0 in 0.723 seconds
```

Unary Operators

Operator	Description
++	Increment
--	Decrement
-	Negative
+	Positive

Unary Operators

```go
func main() {
  x := 1
  x++
  fmt.Println(x)

  x--
  fmt.Println(x)
}
```

Unary Operators

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/flow_control/operator/unary.go"
2
1

[Done] exited with code=0 in 0.641 seconds
```

Comparison Operators

Operator	Description
==	Equal
!=	Not equal
<	Less
<=	Less or equal
>	Greater
>=	Greater or equal

Comparison Operators

```go
func main() {
  a, b := 5, 10
  fmt.Println(a == b)
  fmt.Println(a != b)
  fmt.Println(a > 5, a >= 5)
  fmt.Println(10 < b, 10 <= b)
}
```

Comparison Operators

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/flow_control/operator/comparison.go"
false
true
false true
false true

[Done] exited with code=0 in 0.609 seconds
```

Logical Operators

Operator	Description
&&	Conditional AND
||	Conditional OR
!	Not or Negation

Logical Operators

```go
func main() {
  a, b := 5, 10
  fmt.Println(a < 1 && b == 10)
  fmt.Println(a != 5 || b == 100)
  fmt.Println(!(a > 0))
  fmt.Println(!(a == 1) || (b == 100))
}
```

Logical Operators

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/flow_control/operator/logical.go"
false
false
false
true

[Done] exited with code=0 in 0.596 seconds
```

Conditional Statements

If Expression

Switch Expression

If Expression

- If
- Else if
- Else

```go
func main() {
  var condition bool

  if condition {
    // execute this code
  } else if condition {
    // execute this code
  } else {
    // execute this code
  }
}
```

If Expression

```go
func main() {
  age := 50

  if age >= 0 && age < 18 {
    fmt.Printf("You cannot vote! Please return in %d years!\n", 18-age)
  } else if age == 18 {
    fmt.Println("This is your first vote!")
  } else if age > 18 && age <= 100 {
    fmt.Println("Please vote, it's important!")
  } else {
    fmt.Println("Invalid age!")
  }
}
```

If Expression

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/flow_control/conditional/if.go"
Please vote, it's important!

[Done] exited with code=0 in 0.482 seconds
```

Switch Expression

- Switch
- Case
- Default

```go
func main() {
  var condition interface{}

  switch condition {
  case condition:
    // execute this code
  case condition:
    // execute this code
  default:
    // execute this code
  }
}
```

Switch Expression

```go
func main() {
  language := "golang"

  switch language {
  case "Python":
    fmt.Println("You are learning Python!")

  case "Go", "golang":
    fmt.Println("Good, go for Go!")

  default:
    fmt.Println("Any other programming language is a good start!")
  }
}
```

Switch Expression

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/flow_control/conditional/switch.go"
Good, go for Go!

[Done] exited with code=0 in 0.574 seconds
```

Switch Expression

```go
func main() {
  n := 5

  switch {
  case n%2 == 0:
    fmt.Println("Event number!")
  case n%2 != 0:
    fmt.Println("Odd number!")
  default:
    fmt.Println("Never here!")
  }
}
```

Switch Expression

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/flow_control/conditional/switch.go"
Odd number!

[Done] exited with code=0 in 0.481 seconds
```

Switch Expression

```go
func main() {
  hour := time.Now().Hour()

  switch {
  case hour < 12:
    fmt.Println("Good Morning!")
  case hour < 17:
    fmt.Println("Good Afternoon!")
  default:
    fmt.Println("Good Evening!")
  }
}
```

Switch Expression

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/flow_control/conditional/switch.go"
Good Morning!

[Done] exited with code=0 in 0.543 seconds
```

Loop Statements

For Expression

For Expression

```go
func main() {
  for i := 0; i < 10; i++ {
    fmt.Println(i)
  }
}
```

```go
func main() {
  i := 0
  for i < 10 {
    fmt.Println(i)
    i++
  }
}
```

For Expression

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/flow_control/loop/loop.go"
0
1
2
3
4
5
6
7
8
9

[Done] exited with code=0 in 0.607 seconds
```

For Expression

Break

Continue

Range

For with Break Expression

```go
func main() {
  for i := 0; i < 10; i++ {
    if i == 5 {
      break
    }
    fmt.Println(i)
  }
}
```

For with Break Expression

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/flow_control/loop/loop.go"
0
1
2
3
4

[Done] exited with code=0 in 0.59 seconds
```

For with Continue Expression

```go
func main() {
  for i := 0; i < 10; i++ {
    if i%2 != 0 {
      continue
    }
    fmt.Println(i)
  }
}
```

For with Continue Expression

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/flow_control/loop/loop.go"
0
2
4
6
8

[Done] exited with code=0 in 0.617 seconds
```

For with Range Expression

```go
func main() {
  names := []string{"John", "Paul", "George", "Ringo"}
  for index, name := range names {
    fmt.Println("index", index, "=", name)
  }
}
```

For with Range Expression

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/flow_control/loop/loop.go"
index 0 = John
index 1 = Paul
index 2 = George
index 3 = Ringo

[Done] exited with code=0 in 0.381 seconds
```

COLLECTION TYPES

Collection Types

- Array
- Slice
- Map

Array

- An array is a data type that contains a collection of data of the same type.
- When creating an array, we need to specify the amount of data that the array can hold.
- The amount of data the array can hold cannot change after the array is created.

```go
func main() {
  var numbers [4]int

  fmt.Printf("%v\n", numbers)
  fmt.Printf("%#v\n", numbers)
}
```

Array

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/array/array.go"
[0 0 0 0]
[4]int{0, 0, 0, 0}

[Done] exited with code=0 in 0.565 seconds
```

Array

```go
func main() {
  var a1 = [1]float64{}
  fmt.Printf("%#v\n", a1)

  var a2 = [2]int{-10, 1}
  fmt.Printf("%#v\n", a2)

  var a3 = [3]string{"John", "Paul", "George"}
  fmt.Printf("%#v\n", a3)

  var a4 [4]string
  a4[0] = "x"
  a4[1] = "y"
  fmt.Printf("%#v\n", a4)
}
```

Array

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/data_structure/array/declare.go"
[1]float64{0}
[2]int{-10, 1}
[3]string{"John", "Paul", "George"}
[4]string{"x", "y", "", ""}

[Done] exited with code=0 in 2.048 seconds
```

Array

Array has 2 data :

- Length
- Capacity

```go
func main() {
  // ellipsis operator
  var a5 = [...]int{
    1,
    -2,
    5,
    -10,
    25,
  }
  fmt.Printf("%#v\n", a5)
  fmt.Printf("The length of a5 is %d\n", len(a5))
  fmt.Printf("The capacity of a5 is %d\n", cap(a5))

  a5[0] = 100
  fmt.Printf("%#v\n", a5)
}
```

Array

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/data_structure/array/declare.go"
[5]int{1, -2, 5, -10, 25}
The length of a5 is 5
The capacity of a5 is 5
[5]int{100, -2, 5, -10, 25}

[Done] exited with code=0 in 0.696 seconds
```

Slice

- Slice is a data type that is like an array, as it is a slice of an array.
- The difference is the capacity size that can change.

```go
func main() {
  var cities []string

  fmt.Printf("cities %v\n", cities)
  fmt.Printf("cities %#v\n", cities)
}
```

Slice

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/data_structure/slice/test.go"
cities []
cities []string(nil)

[Done] exited with code=0 in 0.656 seconds
```

Slice

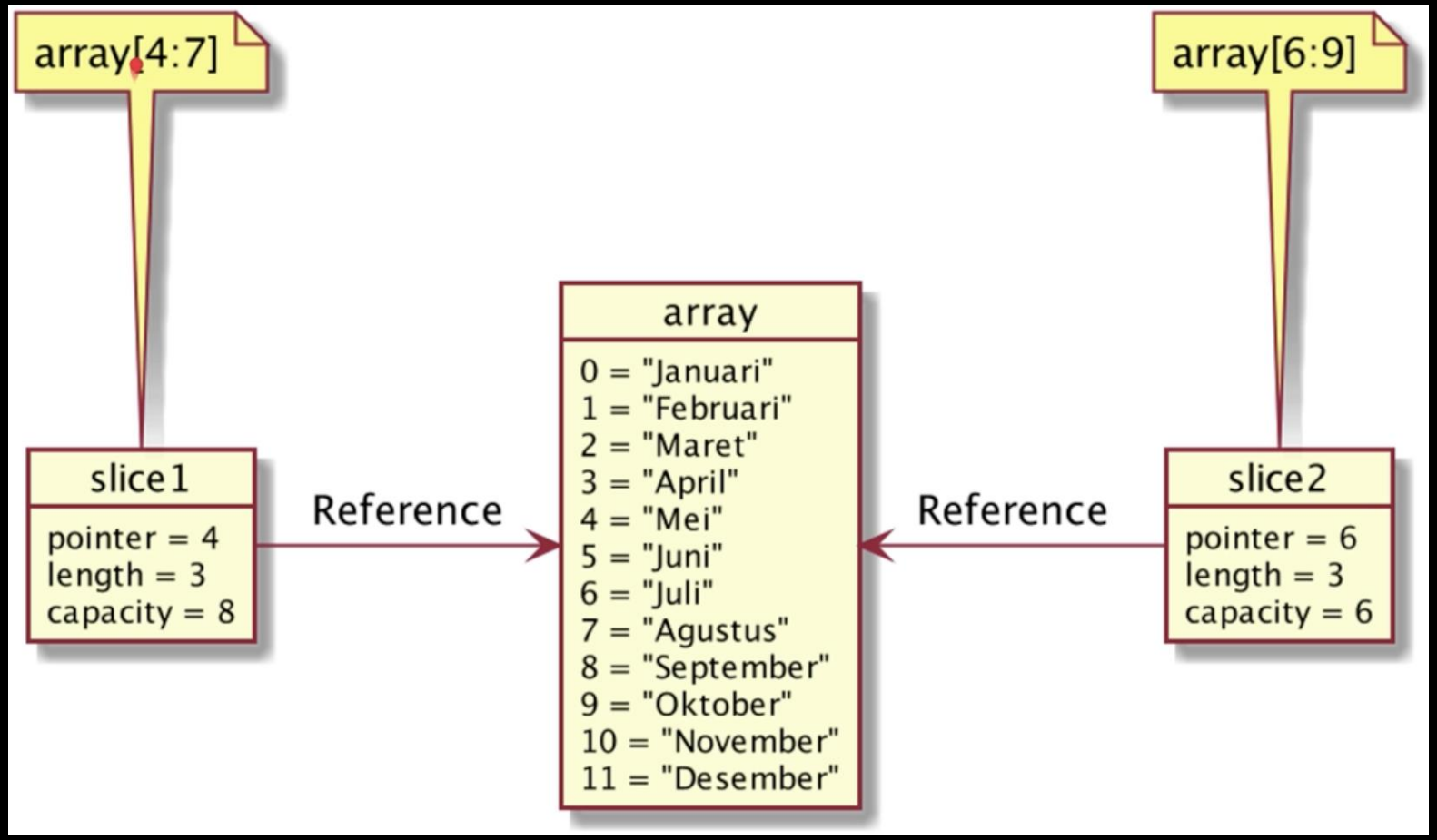
Slice has 3 data

- Pointer
- Length
- Capacity

Slice

Make Slice	Description
array[low:high]	Make a slice of the array from index low to index before high
array[low:]	Make a slice of the array from index low to last index in the array
array[:high]	Make a slice of the array from index 0 to index before high
array[:]	Make a slice of the array from index 0 to last index in the array

Slice


array[4:7]
array[6:9]
array
0 = "Januari"
1 = "Februari"
2 = "Maret"
3 = "April"
4 = "Mei"
5 = "Juni"
6 = "Juli"
7 = "Agustus"
8 = "September"
9 = "Oktober"
10 = "November"
11 = "Desember"
slice1
pointer = 4
length = 3
capacity = 8
Reference
Reference
slice2
pointer = 6
length = 3
capacity = 6

Slice

```go
func main() {
  numbers := []int{2, 3, 4, 5}
  fmt.Println(numbers)

  nums := make([]int, 2)
  fmt.Printf("%#v\n", nums)

  friends := [3]string{"John", "Paul", "George"}
  fmt.Printf("%#v\n", friends)

  myBestFriend := friends[0:1]
  fmt.Println("My best friend is ", myBestFriend)
  fmt.Printf("%#v\n", myBestFriend)

  for index, value := range friends {
    fmt.Printf("index: %v, value: %v\n", index, value)
  }
}
```

Slice

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/slice/declare.go"
[2 3 4 5]
[]int{0, 0}
[3]string{"John", "Paul", "George"}
My best friend is  [John]
[]string{"John"}
index: 0, value: John
index: 1, value: Paul
index: 2, value: George

[Done] exited with code=0 in 0.392 seconds
```

Function Slice

Function Slice	Description
len(slice)	Get the length of the slice
cap(slice)	Get the capacity of the slice
append(slice, data)	Create a new slice by adding data to the last index of the slice (If the capacity is full, it will create an array with 2 times the capacity)
make([]DataType, length, capacity)	Create a new slice
copy(destination, source)	Copy the slice from source to destination

Array vs Slice

Array

- Has a fixed length defined at compile time
- The length of an array is part of its type, defined at compile time and cannot be changed
- By default, an uninitialized array has all elements equal to zero

Slice

- Has a dynamic length (it can shrink or grow)
- The length of a slice is not part of its type, and it belongs to runtime
- An uninitialized slice is equal to nil (its zero value is nil)

Map

- In Array dan Slice, to access the data, we use the Number index that starts from 0.
- Map is another data type that contains the same set of data, but we can specify the type of index data type that we will use.
- Simply put Map is a key-value collection data type, where the keywords are unique and cannot be the same.
- Unlike Array and Slice, the amount of data we put into Map can be as much as possible, as long as the keywords are different.
- If we use the same keyword, then the previous value will automatically be replaced with the new value.

```
{
  key1: value1,
  key2: value2,
  key3: value3,
}
```

Map

```go
func main() {
  var employees map[string]string = map[string]string{
    "name":    "Diva",
    "address": "Bandung",
  }

  fmt.Printf("%#v\n", employees)
  fmt.Println(employees["name"])
  fmt.Println(employees["address"])
  fmt.Printf("Length of employees is %d\n", len(employees))
}
```

Map

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/data_structure/map/test.go"
map[string]string{"address":"Bandung", "name":"Diva"}
Diva
Bandung
Length of employees is 2

[Done] exited with code=0 in 0.51 seconds
```

Function Map

Function Map	Description
len(map)	Get the length of the map
make(map[TypeKey]TypeValue)	Create a new map
delete(map, key)	Delete data of key from the map

EMPTY TYPE

Empty Interface

```go
func ups() interface{} {
  // return 1
  // return true
  return "Ups"
}

func main() {
  data := ups()
  fmt.Println(data)
}
```

Empty Interface

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/empty_type/interface.go"
Ups

[Done] exited with code=0 in 0.561 seconds
```

FUNCTIONS

Functions

- Parameter
- Return Value
- Return Predefined Value

Functions

```go
func helloWorld() {
  fmt.Println("Hello World")
}

func main() {
  helloWorld()
}
```

Functions

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/function/function.go"
Hello World

[Done] exited with code=0 in 0.885 seconds
```

Function Parameter

```go
func helloParameter(firstName string, lastName string) {
  fmt.Println("Hello", firstName, lastName)
}
func main() {
  helloParameter("John", "Doe")
}
```

Function Parameter

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/function/function_parameter.go"
Hello John Doe

[Done] exited with code=0 in 0.648 seconds
```

Function Return Value

```go
func getHelloWorld() (string, string) {
  return "Hello", "World"
}

func main() {
  x, y := getHelloWorld()
  fmt.Println(x, y)
}
```

Function Return Value

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/function/function_return_value.go"
Hello World

[Done] exited with code=0 in 0.645 seconds
```

Function Predefined Value

```go
func getName() (firstName, lastName string) {
  return "Ferdian", "Maulana"
}

func main() {
  x, y := getName()
  fmt.Println(x, y)
}
```

Function Predefined Value

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/function/function_predifined_value.go"
Ferdian Maulana

[Done] exited with code=0 in 0.569 seconds
```

Variadic Function

```go
func sumAll(numbers ...int) (sum int) {
  for _, number := range numbers {
    sum += number
  }
  return
}

func main() {
  total := sumAll(5, 2, 8, 9)
  fmt.Println(total)
}
```

Variadic Function

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/function/variadic_function.go"
24

[Done] exited with code=0 in 0.454 seconds
```

ACCESS MODIFIER

Access Modifier

- Public Property
- Private Property

Access Modifier

```go
func main() {
  fmt.Println(controller.ReadUserList())
}
```

```go
type user struct {
  ID    int    `json:"id"`
  Name  string `json:"name"`
  Email string `json:"email"`
}

func ReadUserList() (users []user) {
  users = make([]user, 2)

  users[0] = user{
    ID:    1,
    Name:  "Ferdian Maulana",
    Email: "ferdian@plabs.id",
  }

  users[1] = user{
    ID:    2,
    Name:  "Diva Dien Alhaq",
    Email: "diva@plabs.id",
  }

  return
}
```

Access Modifier

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/access_modifier/main.go"
[{1 Ferdian Maulana ferdian@plabs.id} {2 Diva Dien Alhaq diva@plabs.id}]

[Done] exited with code=0 in 0.796 seconds
```

DECLARATION TYPES

Declaration Types

- Struct
- Interface

Struct

```go
type User struct {
  ID    int
  Name  string
  Email string
}

func main() {
  var john User

  john.ID = 1
  john.Name = "John"
  john.Email = "john@plabs.id"

  fmt.Printf("%#v\n", john)
}
```

Struct

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/declaration_type/struct.go"
main.User{ID:1, Name:"John", Email:"john@plabs.id"}

[Done] exited with code=0 in 0.825 seconds
```

Interface

- Interface adalah tipe data Abstract, dia tidak memiliki implementasi langsung.
- Sebuah interface berisikan definisi-definisi method.
- Biasanya interface digunakan sebagai kontrak.

METHOD

Method

```go
type User struct {
  ID    int
  Name  string
  Email string
}

func (u User) sayHello() {
  fmt.Println("Hello " + u.Name)
}

func main() {
  var john User

  john.ID = 1
  john.Name = "John"
  john.Email = "john@plabs.id"

  john.sayHello()
  // fmt.Printf("%#v\n", john)
}
```

Method

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/declaration_type/struct.go"
Hello John

[Done] exited with code=0 in 0.578 seconds
```

POINTER

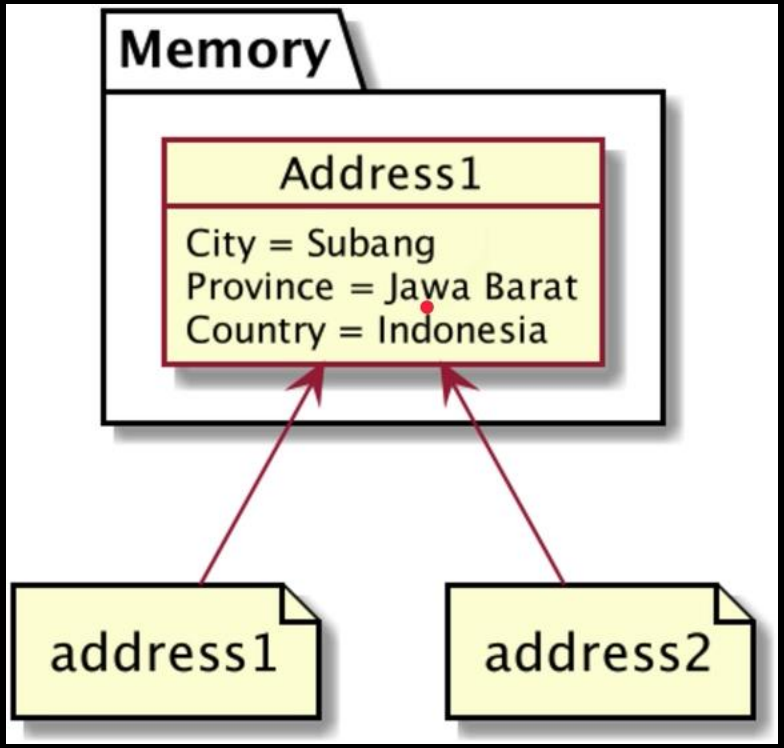
Pointer

```go
type Address struct {
  City     string
  Province string
  Country  string
}

func main() {
  address1 := Address{"Bandung", "Jawa Barat", "Indonesia"}
  address2 := address1

  address2.City = "Jakarta"

  fmt.Println(address1)
  fmt.Println(address2)
}
```

Pointer

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/pointer/pointer.go"
{Bandung Jawa Barat Indonesia}
{Jakarta Jawa Barat Indonesia}

[Done] exited with code=0 in 0.76 seconds
```

Pointer


Memory
Address1
City = Subang
Province = Jawa Barat
Country = Indonesia
address1
address2

Operator Pointer

- Ampersand (&)
- Asterisk (*)

Pointer Operator Ampersand (&)

```go
func main() {
  address1 := Address{"Bandung", "Jawa Barat", "Indonesia"}
  address2 := &address1

  address2.City = "Jakarta"

  fmt.Println(address1)
  fmt.Println(address2)
}
```

Pointer Operator Ampersand (&)

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/pointer/pointer.go"
{Jakarta Jawa Barat Indonesia}
&{Jakarta Jawa Barat Indonesia}

[Done] exited with code=0 in 0.726 seconds
```

Pointer Operator Asterisk (*)

```go
func main() {
  address1 := Address{"Bandung", "Jawa Barat", "Indonesia"}
  var address2 *Address = &address1

  address2.City = "Jakarta"

  fmt.Println(address1)
  fmt.Println(address2)
}
```

DEFER, ERROR, PANIC, RECOVER

Defer

```go
func logging() {
  fmt.Println("Finish running the program")
}

func runApplication() {
  fmt.Println("Run application")
}

func main() {
  defer logging()
  runApplication()
}
```

Defer

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/defer_error/main.go"
Run application
Finish running the program

[Done] exited with code=0 in 0.619 seconds
```

Error & Panic

```go
func divide(nilai, divider int) (int, error) {
  if divider == 0 {
    return 0, errors.New("divider cannot be zero")
  }

  return nilai / divider, nil
}

func main() {
  result, err := divide(10, 0)
  if err != nil {
    panic(err)
  }

  fmt.Println("result", result)
}
```

Error & Panic

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/defer_error/error.go"
panic: divider cannot be zero

goroutine 1 [running]:
main.main()
  /Users/feerdim/plabs/go/assalaam/defer_error/error.go:19 +0x58
exit status 2

[Done] exited with code=1 in 0.601 seconds
```

Recover

```go
func cover() {
  err := recover()
  fmt.Println("Error found :", err)
}

func divide(nilai, divider int) (int, error) {
  if divider == 0 {
    return 0, errors.New("divider cannot be zero")
  }

  return nilai / divider, nil
}

func main() {
  defer cover()

  result, err := divide(10, 0)
  if err != nil {
    panic(err)
  }

  fmt.Println("result", result)
}
```

Recover

```
[Running] go run "/Users/feerdim/plabs/go/assalaam/defer_error/recover.go"
Error found : divider cannot be zero

[Done] exited with code=0 in 0.45 seconds
```