MINAB
IT SOLUTIONS

Backend development Class 04, Series 03

Express

Hahu Jobs

CLASS 04

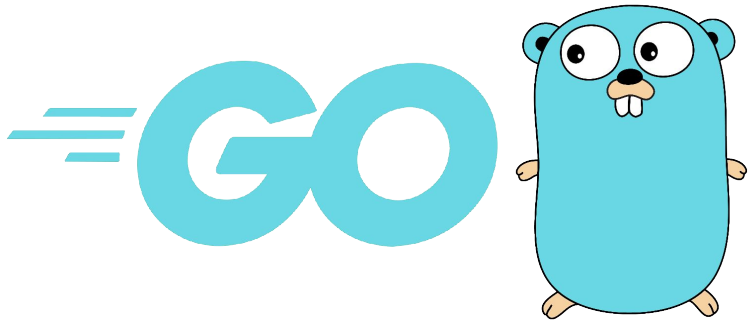
01 Introducing Golang

02 Go Fundamentals

03 What does Go code look like?

Hahu Jobs

Introducing Golang



Golang (Go) is a statically typed, compiled high-level programming language designed at Google

MINAB IT SOLUTIONS

HahuJobs

Introducing Golang

Go was designed at **Google** in **2007** to improve programming productivity in an era of multicore, networked machines and large codebases.
The designers wanted to address criticisms of other languages in use at Google, but keep their useful characteristics

- **Static typing** and run-time efficiency (like C)
- **Readability** and **usability** (like Python)
- **High-performance** networking and **multiprocessing**

Hello Bahir Dar

A Simple Hello Bahir Dar example with Golang

main.go

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, Bahir Dar")
}

```

MINAB IT SOLUTIONS

HahuJobs

A Tour of Go (https://go.dev/tour)

A **Tour of Go** is a great resource for learning the Go programming language in an interactive way.

The tour is divided into modules that cover various Go concepts. Though we will only cover the basic module.

You can access these modules by clicking on "A Tour of Go" at the top left or browsing the table of contents on the top right.

The tour covers the most important features of the language, mainly:

- Basics
- Methods and interfaces
- Generics
- Concurrency

MINAB IT SOLUTIONS

HahuJobs

Packages

Every Go program is made up of packages.

Programs start running in package **main**.

This program is using the packages with import paths "**fmt**" and "**math/rand**".

main.go

```go
package main

import (
    "fmt"
    "math/rand"
)

func main() {
    fmt.Println("My favorite number is", rand.Intn(10))
}
```

MINAB IT SOLUTIONS

HahuJobs

Exported names

In Go, a name is exported if it begins with a capital letter. For example, Pizza is an exported name, as is Pi, which is exported from the math package.

pizza and pi do not start with a capital letter, so they are not exported.

main.go

```go
package main

import (
    "fmt"
    "math"
)

func main() {
    fmt.Println(math.Pi)
}

```

Functions

A **function** can take zero or more arguments.

In this example, add takes two parameters of type int.

Notice that the type comes after the variable name.

main.go

```go
package main

import "fmt"

func add(x int, y int) int {
    return x + y
}

func main() {
    fmt.Println(add(42, 13))
}

```

Functions continued

When two or more consecutive named function parameters share a type, you can omit the type from all but the last.

In this example, we shortened

main.go

```go
package main

import "fmt"

func add(x, y int) int {
    return x + y
}

func main() {
    fmt.Println(add(42, 13))
}

```

MINAB IT SOLUTIONS

HahuJobs

Multiple results

A function can return any number of results.

The swap function returns two strings.

main.go

```go
package main

import "fmt"

func swap(x, y string) (string, string) {
    return y, x
}

func main() {
    a, b := swap("hello", "world")
    fmt.Println(a, b)
}
```

MINAB IT SOLUTIONS

HahuJobs

Named return values

Go's return values may be named. If so, they are treated as variables defined at the top of the function.

These names should be used to document the meaning of the return values.

A return statement without arguments returns the named return values. This is known as a "**naked**" return.

main.go

```go
package main

import "fmt"

func split(sum int) (x, y int) {
    x = sum * 4 / 9
    y = sum - x
    return
}

func main() {
    fmt.Println(split(17))
}

```

Variables

The **var** statement declares a list of variables; as in function argument lists, the type is last.

A **var** statement can be at package or function level. We see both in this example.

main.go

```
1 package main
2
3 import "fmt"
4
5 var c, python, java bool
6
7 func main() {
8     var i int
9     fmt.Println(i, c, python, java)
10 }
11
```

MINAB IT SOLUTIONS

HahuJobs

Variables with initializers

A var declaration can include initializers, one per variable.

If an initializer is present, the type can be omitted; the variable will take the type of the initializer.

main.go

```go
package main

import "fmt"

var i, j int = 1, 2

func main() {
    var c, python, java = true, false, "no!"
    fmt.Println(i, j, c, python, java)
}

```

Short variable declarations

Inside a function, the := short assignment statement can be used in place of a var declaration with implicit type.

Outside a function, every statement begins with a keyword (**var**, **func**, and so on) and so the := construct is not available.

main.go

```go
package main

import "fmt"

func main() {
    var i, j int = 1, 2
    k := 3
    c, python, java := true, false, "no!"

    fmt.Println(i, j, k, c, python, java)
}

```

MINAB IT SOLUTIONS

HahuJobs

Basic types

Go's basic types are

- bool
- string
- int int8 int16 int32 int64
- uint uint8 uint16 uint32 uint64 uintptr
- byte // alias for uint8
- rune // alias for int32 & represents a Unicode code point
- float32 float64
- complex64 complex128

main.go

```go
package main

import (
    "fmt"
    "math/cmplx"
)

var (
    ToBe   bool       = false
    MaxInt uint64     = 1<<64 - 1
    z      complex128 = cmplx.Sqrt(-5 + 12i)
)

func main() {
    fmt.Printf("Type: %T Value: %v\n", ToBe, ToBe)
    fmt.Printf("Type: %T Value: %v\n", MaxInt, MaxInt)
    fmt.Printf("Type: %T Value: %v\n", z, z)
}

```

Zero values

Variables declared without an explicit initial value are given their zero value.

The zero value is:

0 for numeric types,

false for the boolean type, and

"" (the empty string) for strings.

main.go

```go
package main

import "fmt"

func main() {
    var i int
    var f float64
    var b bool
    var s string
    fmt.Printf("%v %v %v %q\n", i, f, b, s)
}

```

Type conversions

The expression **T(v)** converts the value **v** to the type **T**.

Unlike in C, in Go assignment between items of different type requires an explicit conversion. Try removing the **float64** or **uint** conversions in the example and see what happens.

main.go

```go
package main

import (
    "fmt"
    "math"
)

func main() {
    var x, y int = 3, 4
    var f float64 = math.Sqrt(float64(x*x + y*y))
    var z uint = uint(f)
    fmt.Println(x, y, z)
}

```

Type inference

When declaring a variable without specifying an explicit type (either by using the := syntax or var = expression syntax), the variable's type is inferred from the value on the right hand side.

var i int
j := i // j is an int

MINAB IT SOLUTIONS

HahuJobs

Constants

Constants are declared like variables, but with the **const** keyword.
Constants can be **character**, **string**, **boolean**, or **numeric** values.
Constants cannot be declared using the **:=** syntax.

main.go

```go
package main

import "fmt"

const Pi = 3.14

func main() {
    const World = "世界"
    fmt.Println("Hello", World)
    fmt.Println("Happy", Pi, "Day")

    const Truth = true
    fmt.Println("Go rules?", Truth)
}

```

Numeric Constants

Numeric constants are high-precision values.

An untyped constant takes the type needed by its context.

main.go

```go
package main

import "fmt"

const (
    // Create a huge number by shifting a 1 bit left 100 places.
    // In other words, the binary number that is 1 followed by 100 zeroes.
    Big = 1 << 100
    // Shift it right again 99 places, so we end up with 1<<1, or 2.
    Small = Big >> 99
)

func needInt(x int) int { return x*10 + 1 }
func needFloat(x float64) float64 {
    return x * 0.1
}

func main() {
    fmt.Println(needInt(Small))
    fmt.Println(needFloat(Small))
    fmt.Println(needFloat(Big))
}

```

Assignment

- Complete “**a tour of Go**” basic module
- Google and install **Postgres** Database on your computer
- Google and install **Golang** on your computer
- Google and install **Docker** on your computer
- Google and and try to install **hasura** with docker on your computer
- Make your **Rick and Morty** website responsive (**Mobile** and **Desktop**)
- Write a simple calculator HTTP server with golang that accepts JSON payload with any two numbers and return JSON with the result
    - Your HTTP server should have the following **http methods** & **endpoints**
        - GET /add (adds two argument arg1 & arg2 and return result)
        - GET /sub (subtract arg2 from arg1 and return result)
        - POST /mul (multiply two argument arg1 & arg2 and return result)
        - POST /div (divide arg2 from arg1 and return result)

MINAB IT SOLUTIONS

HahuJobs

Assignment

- examples
- http://localhost:8000/add
  payload: {“arg1”:10, “arg2”:5}
  output result: {“result”: 15}
- http://localhost:8000/mul
  payload: {“arg1”:2, “arg2”:50}
  output result: {“result”: 100}
- http://localhost:8000/sub
  payload: {“arg1”:30, “arg2”:-100}
  output result: {“result”: 70}

Note: Download and use Postman to test your http server

MINAB IT SOLUTIONS

HahuJobs

MINAB
IT SOLUTIONS

Thank you!

Hahu Jobs

ለሀገር ልጅ በሀገር ልጅ !

Michael Sahlu
michael.sahlu@hahu.jobs