GO Programming language

Overview:

* Introduction
* Hello world
* Variables
* Functions
* Conditionals
* Loops
* Arrays and Slices
* Maps
* Structs
* Introduction to Concurrency

Introduction:

* Initial release 2009
* It has only **25** inbuilt keywords
* Cross platform
* It is a compiled language –> Code we write, compiled down into machine code before it runs. So that the operating system understands the code and it does not need a runtime interpreter.
* It is a strongly typed language (Sticking with types) – If we try to add an integer with a floating point number, the compiler will throw a type mismatch error.
* It supports type inference (automatic detection of data type)

a := 5.5 – In GO language –> Here 5.5 is a float. And it is going to create the variable (a) as a floating point variable

float a = 5.5 – In C/C++/JAVA

Creating Hello world program

Package main

import “fmt”

import “runtime”

func main(){

fmt.println(“Hello world from ”, runtime.GOOS)

}

**OUTPUT**:

Hello world from windows

Explanation:

package main – This is required if it is a standalone application. But not required for shared library (e.g JAR)

import “fmt” – These are packages

import “runtime” – These are packages

func – GO keyword

main – It is the main function

All functions have parenthesis

The function code goes into the curly braces

Variables declaration

Package level variable

Rules

* Variables may or may not be used in the program.
* If we define variable outside a function (at a package level), we need to use var keyword.

Declaring bunch of variables at package level, we can do like below

Declaration

Var(

Name string //variable names on the left and type on the right

Course string //variable names on the left and type on the right

Module float //variable names on the left and type on the right

)

We can either declare types like above or the compiler will automatically detect the data type for you.

The below code is also legal for declaring the variable

Other way of declaration

Var(

Name = “Vishnu”

Course = “Go Programming”

Float = 1.4

)

Method level variable

Shorthand declare and initialize ‘:=’

Rules

Variables declared at the function level must be used

Example 1

name := “Vishnu”

course := “Go Programming”

module := 1.4

fmt.Println(“Name is ”,name)

fmt.Println(“Course is ”,course)

Output

Throws error. Because all the short hand variables must be used.

Example 2

a := 10.25

b := 5

c := a + b // Go will throw mismatch typed exception here. Because Go can’t sum int and float. So we need to do the Type conversion here

c := int(a) + b //Valid

OUTPUT:

15

Pointers

Name := “Vishnu”

ptr := &name –> References a pointer –> Gives the address

\*ptr –> De-references a pointer –> Get the value back

& - References a pointer and \* - De-references a pointer

Pass by value

It will take a copy of the value (same value having different address) and the original value won’t get affected even if we try to change

Example:

Func main(){

Course := “Go programming language”

fmt.println(“Course name ” , course)

changeCourse(Course)

fmt.println(“Course name ” , course)

}

Func changeCourse(course string) string{

course = “Java programming language” // We are trying to change the value here

fmt.println(course)

return course

}

**OUTPUT**:

GO programming language

Java programming language

GO programming language

Pass by reference

Changes to the variables in the functions will affect the originals and no copy of the value is made

func square(x \*float64) {

\*x = \*x \* \*x

}

func main() {

x := 1.5

square(&x)

fmt.Println(x)

}

**OUTPUT**:

2.25

Functions:

Functions are reusable components

**GO Function signature**

func toUpperCase(text,value string) (string,string){

return text, value

}

**func** – keyword to tell that, it is the function to the compiler

**toUpperCase** – function name

**(text, value string)** – Here **text** and **value** is an argument and **string** is a type of an argument

**return** – exists function

**(string,string) (multiple return values) -** return type is **string**

Example program

package main

import (

"fmt"

"strings"

)

func main(){

module := "function basics"

author := "Vishnu"

fmt.Println(module,author)

fmt.Println(converter(module,author))

}

func converter(module, author string) (string,string){

module = strings.Title(module)

author = strings.ToUpper(author)

return module,author

}

**OUTPUT**

**function basics Vishnu**

**Function Basics VISHNU**

Named returns

Go functions has the ability to name the return values

package main

import "fmt"

func main() {

message,alternate := greet("Hello","Vishnu")

fmt.Println(message)

fmt.Println(alternate)

}

func greet(greet string, name string)( message string, alternate string){

message = greet + " " + name

alternate = "Hey "+ name

return

}

And if we want to get only one return value from the multiple values, we can do like below.

\_ , alternate := greet("Hello","Vishnu")

Variadic functions

Variadic functions are used, when we are not sure that how many values are going to be passed to a function

Example 1

package main

import "fmt"

func main() {

bestEmployee := getBestEmployee(100,101,102,103,104,105,106)

fmt.Println(bestEmployee)

}

func getBestEmployee(employeesId ...int)int{

first := employeesId[0]

return first

}

employeesId ...int – these ellipses define that it is a variadic functions

**OUTPUT**

100

Example 2

package main

import "fmt"

func main() {

message , alternate := greet("Vishnu","Hello","Yo")

fmt.Println(message)

fmt.Println(alternate)

}

func greet(name string, greet ...string)(message string, alternate string){

message = greet[1] + " " + name

alternate = "Hey "+ name

return

}

**OUTPUT**

Yo Vishnu

Hey Vishnu

Function types (Passing a function as an argument)

package main

import "fmt"

func myfn1(i string) {

fmt.Println(i)

}

func myfunc2(firstName string, lastName string) string {

return "Hello "+ firstName + " " + lastName + "!"

}

// Here we are giving function reference name as ‘do’

func test(do func(string), val string) {

do(val) // This will call the myfn1

}

// Here we are giving function reference name as ‘t’

func test1(t func(string,string) string, fname string, lname string) string {

opt := t(fname, lname) // This will call myfunc2

return opt

}

func main() {

test(myfn1, "Vishnu") // we are passing the function name here along with the matching arguments to that function

greet := test1(myfunc2, "Vishnu","KP") // we are passing the function name here along with the matching arguments to that function

fmt.Println(greet)

}

**OUTPUT**

Vishnu

Hello Vishnu KP!