

GoLang

* Founder :  [Robert Griesemer](https://en.wikipedia.org/wiki/Robert_Griesemer), [Rob Pike](https://en.wikipedia.org/wiki/Rob_Pike), and [Ken Thompson](https://en.wikipedia.org/wiki/Ken_Thompson)
* Developed by Google in 2007.
* Open Sourced in 2009.
* Go is Statically typed, Compiled high-level Programming language.

**Note : I’m creating these notes as considering the reader have technical background. Reader have a basic knowledge of any other programming language.**

1. What is Golang used for ?

* Go is popular in Cloud-based application or Server-side app.
* DevOps & website reliability automation are two well know ways to utilize Go.
* A lot of command-line tools have been written in Go.
* Go is used in AI & Data Science.

0.1 Advantage

* Fast
* Easy to learn
* Well scaled
* Comprehensive Programming Tool
* Strong Community Support (Google)

0.2 Disadvantage

* Young Language

0.3 Features

* Light weight (Goroutine take 8kb)
* Simplicity
* Concurrency
* Garbage Collection
* Cross Platform Support
* Fast Compile Time
* Strong Typing
  1. Installation of GoLang
* Go to <https://go.dev/doc/install> & download the installer according to Operating System.
* After installation run the command **go version** on command prompt for checking the successful installation of Go.

0.5 Default Directory Structure

* After installation it creates **go** directory in our **Root** directory.
* To know the path of directory run **go env GOPATH** command.
* In my case it shows **C:\Users\Prajwal\go** & this path also known as **GOPATH**.
* Following is a Default structure of Directory :

go/

bin/

( executable binaries )

pkg/

( compiled package file )

src/

( Myproject )

* In starting there are only 2 folders are present in directory first bin & second pkg, src folder is created by user.
* According to go standards it says do all coding under Go Workspace means inside go directory.
* But according to our convenient we can create our project folder outside GOPATH, but we have to perform some extra steps.

0.6 Go Module

* Go module helps to communicate with GOPATH, it imports the required packages from GOPATH.
* Go Module simplify the management of dependencies & project structure.
* Go Module allows you to create & manage the projects outside the GOPATH.
* To initialize the folder as a Module we have to run **go mod init MODULE\_NAME** command at our folder path in terminal.
* After running command it creates **go.mod** file.
* go.mod file contains the information about our project, including its name & dependencies.
* This process is mandatory when we create our project or folder out side the GOPATH.

0.7 First Program

package main

import “fmt”

func main(){

fmt.println(“Hello World!!!”)

}

0.8 Packages

* Go use Packages instead of classes.
* Each go file must belong to some package.
* Syntax to define package :

**package PACKAGE\_NAME**

* The **main** package is a special package in go. An executable program must contain the **main** package.
* Go uses relative imports to bring packages into current file.
* We can import packages using **import** keyword.
* **main** function is an entry point of our executable program. It should be under the main package.

0.9 Variables

* With the help of **var** & **const** keyword we can initialize or declare the variables.
* There are few ways to create variables.

Case 1 :

var variable\_name data\_type = value

var str1 string = “String 1”

Case 2 :

var variable\_name

var num

Case 3 :

variable\_name := value

pi := 3.14

* In case 1 we specify the datatype of variable. We can’t change datatype of variable later.
* In case 2 we only specify the variable name, we can change the datatype of variable according to our data.
* In case 3 we use **:=** operator instead of var keyword. According to data it will initialize the datatype to variable.
* We can create Constant variables with const keyword & normal variables with var.
* We can’t change the value after declaring the constant variable.
* Ex :

Const pi = 3.14

* When we want to export any variable or function then its first letter of variable must be **capital**. We can export it in various packages & files.
* If variable name is in lowercase then that variable is accessible in only that file. We can’t export it for external use.
* Ex :

var Public\_variable

var private\_variable

**Questions**

1. What is the syntax for declaring a variable in Go?
2. How do you declare multiple variables in a single line in Go?
3. What is the zero value in Go, and how does it relate to variables?
4. How do you declare a constant in Go?
5. What is the short variable declaration, and when should it be used?
6. How do you swap values of two variables in Go?
7. Can you reassign a value to a variable declared with the := syntax? Explain.
8. What is the scope of a variable in Go?
9. How do you declare a variable without assigning an initial value?
10. What happens if you try to use a variable that has not been initialized?
11. Data Types In Go

// Boolean

var isActive bool = true

// Integer

var age int = 30

// Float

var score float64 = 89.7

// Complex

var complexNum complex128 = complex(1.2, 3.4)

// String

var name string = "Hello World!!"

// Array

var numbers [5]int = [5]int{1, 2, 3, 4, 5}

// Slice

var primes []int = []int{2, 3, 5, 7, 11}

// Struct

type Person struct {

Name string

Age int

}

// Pointer

var ptr \*int = &age

// Function

func add(a int, b int) int {

return a + b

}

// Interface

type Shape interface {

Area() float64

}

// Map

var phoneBook map[string]string = map[string]string{

"John": "123-456-7890",

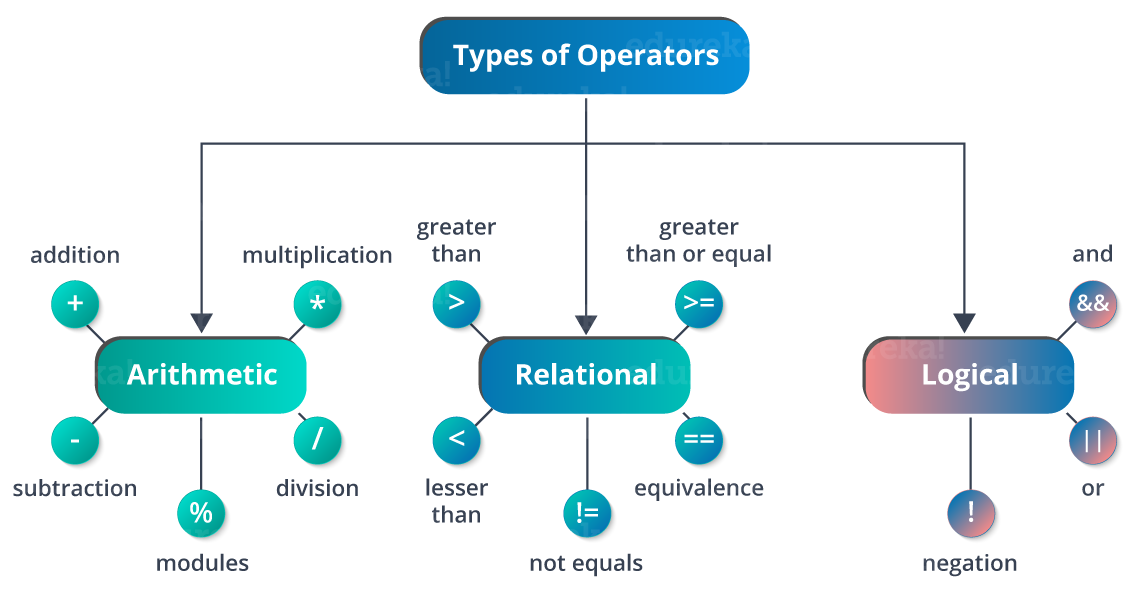
"Jane": "987-654-3210",

}

// Channel

var ch chan int = make(chan int)

* 1. Operators in Go



1.2 Input & Output

* We can write the output with three methods of **fmt** package.
* Println :
  + It prints the statement and add space before the variable printing.
  + After printing the statement set the cursor on next line.
* Print :
  + It can’t add space & can’t set the cursor on next line.
  + It only print the statements.
* Printf :
  + It works like printf() function of C programming.
  + It use format specifiers for printing the statements.
* There are some several ways to take input according to user data.
* There are three **fmt** package methods define for taking input.
* Scan :
  + It is use to take single value at a time.
  + Scan scans text read from standard input, storing successive space-separated values into successive arguments.(referred form website)
* Scanln :
  + Scanln is similar to Scan, but stops scanning at a newline and after the final item there must be a newline or EOF.
* Scanf :
  + It is similar to Scanf() function of C Programming.
  + It use format specifier to take input according to data.
* These functions are take only one word string. We want to use BufferReader for accepting long string.
* We can take non string data easily with these functions.

**Questions**

1. How do you print output to the console in Go?

2. How do you read input from the console in Go?

3. What is the fmt package, and what are its common functions for I/O operations?

4. How do you format strings using the fmt package?

5. What is the difference between fmt.Print, fmt.Println, and fmt.Printf?

6. How do you handle errors when reading input in Go?

7. How do you read an entire line of input from the console?

* 1. Decision Making in Go
* If – Else Statement :
  + It is similar to other programming language.
  + Conditions are not enclosed in parenthesis.
  + Opening curly brace { is compulsory on same line of if statement.
  + Starting of else block is compulsory on closing curly brace } of if statement.
  + Example1 :

if age >= 16 {

fmt.Println("Adult")

} else {

fmt.Println("Not an adult")

}

* Example2 :

if age >= 16 {

fmt.Println("in school")

} else if age >= 18 {

fmt.Println("in college")

} else {

fmt.Println("probably dead")

}

* Switch Statement :
  + It is also similar to other programing language.
  + We can switch the case based on single or multiple values & expressions also.
  + There is no use of **break** in go.
  + Example1 :

// compare with single value

var day = 2

switch day {

case 1 :

fmt.Println(“Monday”)

case 2 :

fmt.Println(“Tuesday”)

case 3 :

fmt.Println(“Wednesday”)

default :

fmt.Println(“Unkown Day”)

}

* + Example 2:

var month = “Mar”

switch month {

case “Oct”, “Nov”, “Dec”, “Jan” :

fmt.Println(“Winter”)

case “Feb”, “Mar”, “Apri”, “May” :

fmt.Println(“Summer”)

default :

fmt.Println(“Rainy”)

}

* + Example 3 :

var temp = 25

switch {

case temp < 0:

fmt.Println(“Freez”)

case temp > 0 && temp < 20:

fmt.Println(“Cold”)

case temp > 20 && temp <30:

fmt.Println(“Warm”)

default :

fmt.Println(“Hot”)

}

**Questions**

 **If Statement:**

* How do you write a basic if statement in Go?
* Can you use an initialization statement with an if statement in Go? Provide an example.

 **If-Else Statement:**

* How do you write an if-else statement in Go?
* What is the purpose of an else if clause, and how is it used?

 **Switch Statement:**

* How do you write a basic switch statement in Go?
* Can you use multiple expressions in a case clause? Provide an example.
* What is the default clause in a switch statement?

 **Switch with Initialization:**

* How do you use an initialization statement in a switch statement? Provide an example.

 **Switch with Types:**

* How do you perform a type switch in Go? Provide an example.

 **Nested Decision Statements:**

* How can you nest if and switch statements in Go? Provide an example.

 **Boolean Operators:**

* What are the common boolean operators used in decision statements in Go? Provide examples of &&, ||, and !.
  1. Loops in Go
* There is only one loop in Go i.e. For loop.
* For loop is similar to other programming language.
* for keyword is used.
* There are 3 sections first counter initialization, second counter condition, third counter increment/decrement separated with **;** .
* There are two ways to implement for loop.
* Example 1 :

for i := 1; i <= 10; i++ {

fmt.Println( i )

}

* We can elaborate for loop also like other programming language. Means we can write initialization part & increment/decrement part separately.
* Example 2 :

X := 1

for x <= 5 {

fmt.Println( x )

x++

}

* Go supports **break** & **continue** also.
* Break :
  + **break** is used to break the loop at specific condition.
* Continue :
  + **continue** is used to break current iteration & move on to next iteration.
* There is a new concept of **range** keyword is used with for loop.
* Range :
  + **range** keyword is used to travel or iterate on elements of collections like array, slice, map & string.
  + **range** keyword return the **index** & **value** of that index in two separate variables.
  + Example 1 :

var str = “Hello Prajwal”

for index, char := range arr {

fmt.Println("Index :", index, " Character :", char)

}

* + Example 2 :

var arr = [5]int{10, 20, 30, 40, 50}

for index, value := range arr {

fmt.Println("Index :", index, " Value :", value)

}

**Questions**

 **For Loop:**

* How do you write a basic for loop in Go?
* Can you provide an example of a for loop that iterates from 1 to 10?

 **Range Loop:**

* What is the range keyword, and how is it used with loops in Go?
* How do you iterate over elements in a slice using a range loop?

 **Infinite Loop:**

* How do you create an infinite loop in Go?
* How can you break out of an infinite loop?

 **While Loop:**

* How do you emulate a while loop in Go using a for loop?
* Provide an example of a for loop that functions like a while loop.

 **Nested Loops:**

* How do you write nested loops in Go?
* Provide an example of a nested loop.

 **Break Statement:**

* What is the break statement, and how is it used in loops?
* Provide an example of using the break statement in a loop.

 **Continue Statement:**

* What is the continue statement, and how is it used in loops?
* Provide an example of using the continue statement in a loop.

 **Labels in Loops:**

* How do you use labels with loops in Go?
* Provide an example of breaking out of an outer loop using a label.

 **Loop Control with Conditionals:**

* How do you control loop execution using if statements inside loops?
* Provide an example where an if statement is used inside a loop to filter elements.

 **Loop with Initialization and Post Statements:**

* How do you use initialization and post statements in a for loop?
* Provide an example of a for loop that includes initialization and post statements.
  1. Functions
* Functions are declared with **func** keyword, followed by name, parameter (if any), return type or variable (if any) and the function body.
* Syntax :

func Function\_name(Parameters) (variable & Return\_type){

//body

}

* Examples :

// function without parameter & return type

func f1(){

fmt.Println(“Function 1”)

}

// function with parameter & return type

func f2(n1, n2 int) int {

return n1+n2

}

// function with parameter & variable return type

func f3(n1, n2 int) (ans int) {

ans= n1+n2

return

}

* We can give common datatype in input parameter if there is same datatype variable. We can give it separately also.

func f2(n1 int, n2 int)

* In example 3, we declare the return variable & return type of function. **ans** is return variable & **int** is return type. If we declare the return variable then it is not necessary to mention it again in return statement, it will automatically return the value of return variable i.e. **ans**.
* Opening curly bracket **{** must be on same line of function declaration. It is declared structure of GoLang.

**Questions**

 **Function Declaration:**

* How do you declare a function in Go?
* What is the syntax for defining a function that returns a value in Go?

 **Multiple Return Values:**

* How can a function return multiple values in Go?
* Provide an example of a function that returns two integers.

 **Named Return Values:**

* What are named return values, and how are they used in Go functions?
* Give an example of a function using named return values.

 **Variadic Functions:**

* What is a variadic function in Go?
* How do you define and call a variadic function?

 **Function as a Value:**

* How can you assign a function to a variable in Go?
* Provide an example where a function is passed as an argument to another function.

 **Anonymous Functions:**

* What is an anonymous function in Go?
* How do you define and invoke an anonymous function?

 **Closures:**

* What is a closure in Go?
* Provide an example of a closure in Go.

 **Defer Statement:**

* What is the defer statement, and how is it used in functions?
* Give an example demonstrating the use of defer in a function.

 **Recursive Functions:**

* What is recursion, and how do you write a recursive function in Go?
* Provide an example of a recursive function.

 **Method vs. Function:**

* What is the difference between a method and a function in Go?
* How do you define a method on a type in Go?
  1. Array
* Array is a Collection of Similar data.
* The concept of array is similar to other programming language.
* Following are basics of Array in Go :
* Declaring an Array :
  + You declare an array by specifying its length and the type of elements it will hold.
  + var arr [5]int // an array of 5 integers, initialized to zero
  + arr := [3]string{"Go", "Python", "Java"} // an array of 3 strings
  + arr := [...]int{1, 2, 3, 4, 5} // array length is inferred from the number of elements
* Accessing Elements :
  + Elements in an array can be accessed using the index, which starts at 0.

arr := [3]int{10, 20, 30}

fmt.Println(arr[0]) // Output: 10

fmt.Println(arr[2]) // Output: 30

* Modifying Elements :
  + You can modify elements in an array by assigning a new value to a specific index.

arr[1] = 25

fmt.Println(arr) // Output: [10 25 30]

* Length of an Array :
  + Use the len function to find the number of elements in the array.

arr := [5]int{1, 2, 3, 4, 5}

fmt.Println(len(arr)) // Output: 5

* Iterating Over an Array :
  + You can iterate over an array using a for loop or a for range loop.

arr := [3]int{10, 20, 30}

// Traditional for loop

for i := 0; i < len(arr); i++ {

fmt.Println(arr[i])

}

// for range loop

for i, v := range arr {

fmt.Printf("Index %d, Value %d\n", i, v)

}

* Copying Arrays :
  + In Go, arrays are copied by value, meaning when you assign one array to another, a copy of the original array is made.

arr1 := [3]int{1, 2, 3} arr2 := arr1 // arr2 is a copy of arr1

* Multidimensional Arrays :
  + Arrays can be multidimensional, like a 2D array (array of arrays).

var matrix [2][3]int // a 2x3 array

matrix = [2][3]int{

{1, 2, 3},

{4, 5, 6},

}

* Slices from Arrays :
  + You can create a slice from an array. A slice is a dynamically-sized view into the array.

arr := [5]int{1, 2, 3, 4, 5}

slice := arr[1:4] // creates a slice containing {2, 3, 4}

fmt.Println(slice) // Output: [2 3 4]

**Questions**

 **Array Declaration:**

* How do you declare an array in Go?
* Provide an example of declaring an array of integers with a length of 5.

 **Array Initialization:**

* How do you initialize an array in Go?
* What is the zero value of an array element in Go?

 **Accessing Array Elements:**

* How do you access and modify elements in an array?
* Provide an example of accessing the third element of an array and modifying it.

 **Array Length:**

* How do you get the length of an array in Go?
* Write a short code snippet to print the length of an array.

 **Iterating Over an Array:**

* How do you iterate over all elements in an array using a for loop?
* Provide an example of using a range loop to iterate over an array.

 **Multidimensional Arrays:**

* How do you declare and initialize a two-dimensional array in Go?
* Provide an example of accessing an element in a two-dimensional array.

 **Array Slicing:**

* Can you create a slice from an array in Go? How?
* Provide an example of creating a slice from an array.
  1. Slices
* Slices is a better version of Array.
* Slices is a **flexible** & **dynamic** data structure that provides more powerful alternative to array.
* There are two way to initialize Slice.
* Example 1 :
  + number := [ ]int {1,2,3,4,5}
  + In this syntax the curly braces are compulsory, if we want to create empty slice then,

number := [ ]int { }

* Example 2 :
  + make() function is used to create a slice with specific length & capacity.
  + It contains 3 parameters first is data type, second is size(length) , third is capacity.
  + Third parameter is a optional parameter.
  + number = make([ ] int, 3, 5)
  + 3 is a length of slice & 5 is capacity of slice.
  + make() function provides the facility of creating slice length & capacity with user choice.
* If we create a slice with make() function and if we insert the data out of its capacity then slice automatically double its capacity.
* In example we have a slice with 3 capacity & if we insert the 4 values if slice then slice automatically double its capacity i.e 6
* Following are some functions & operations on slice :
  + append( ) : To add the elements in Slice. It has 2 parameters first is slice name, second is elements which we want to add & It returns the updated slice.

s := []int{1, 2, 3}

s = append(s, 4, 5) // s is now {1, 2, 3, 4, 5}

* + len() & cap() :
    - Use the len function to get the number of elements in the slice.
    - Use the cap function to get the capacity of the slice (the underlying array size).

s := make([]int, 3, 5)

fmt.Println(len(s)) // Output: 3 fmt.Println(cap(s)) // Output: 5

* + copy( ) : To copy elements from one slice to another.

src := []int{1, 2, 3}

dest := make([]int, len(src))

copy(dest, src)

// dest now contains {1, 2, 3}

* + Slicing a Slice :

s := []int{1, 2, 3, 4, 5}

subSlice := s[1:4] // subSlice contains {2, 3, 4}

* Nil slice & Empty slice :

var nilSlice []int // nil slice

emptySlice := []int{} // empty slice

* Reslicing :
  + You can re-slice a slice to narrow or expand the range of accessible elements, as long as it stays within the original capacity.

s := []int{1, 2, 3, 4, 5}

s = s[:4] // now s is {1, 2, 3, 4}

s = s[2:] // now s is {3, 4}

* Multidimensional Slices :
  + Slices can be composed of other slices to create multidimensional slices.

matrix := [][]int{

{1, 2, 3},

{4, 5, 6},

{7, 8, 9},

}

* Iterating Over slices :
  + You can iterate over slices using a for loop or for range loop.

s := []int{1, 2, 3, 4, 5}

for i := 0; i < len(s); i++ {

fmt.Println(s[i])

}

for i, v := range s {

fmt.Printf("Index %d, Value %d\n", i, v)

}

**Questions**

 **Slice Declaration:**

* How do you declare a slice in Go?
* What is the difference between an array and a slice?

 **Slice Initialization:**

* How do you initialize a slice in Go?
* Provide an example of creating a slice with predefined values.

 **Length and Capacity:**

* How do you find the length and capacity of a slice in Go?
* What is the difference between the length and capacity of a slice?

 **Slicing an Array:**

* How do you create a slice from an array?
* Provide an example of creating a slice from an array and modifying the slice.

 **Appending to a Slice:**

* How do you append elements to a slice in Go?
* What happens to the capacity of a slice when you append elements beyond its capacity?

 **Copying Slices:**

* How do you copy one slice to another in Go?
* Provide an example of using the copy function to copy elements from one slice to another.

 **Slice with make:**

* How do you create a slice using the make function?
* What are the parameters required by the make function for slices?

 **Iterating Over a Slice:**

* How do you iterate over the elements of a slice using a for loop?
* Provide an example of using a range loop to iterate over a slice.

 **Reslicing:**

* What is reslicing in Go?
* Provide an example of reslicing an existing slice.

 **Multi-Dimensional Slices:**

* How do you create a multi-dimensional slice in Go?
* Provide an example of a two-dimensional slice and how to access its elements.
  1. Maps
* Map data structure provides an unordered collection of key-value pair.
* It is similar to dictionaries in python, hash map in cpp.
* It is used to associate values with keys & allows to retrieve the values based on keys.
* Keys are unique.
* Creating a Map :
  + You can create a map using the make function or by using a map literal

// Using make

m := make(map[string]int) // creates an empty map with string keys and int values

// Using a map literal

m := map[string]int{

"apple": 2,

"orange": 3,

}

* Adding or Updating Elements :
  + You can add new key-value pairs or update existing ones by assigning a value to a key.

m := make(map[string]int)

m["apple"] = 2

m["orange"] = 3

m["apple"] = 5 // Updates the value

* Accessing Elements :
  + You can access a value in a map using the key. If the key does not exist, it returns the zero value for the value type.

fmt.Println(m["apple"]) // Output: 5

fmt.Println(m["banana"]) // Output: 0 (default //int value, because "banana" does not exist)

* + To check if a key exists, you can use the two-value assignment.

value, exists := m["apple"]

if exists {

fmt.Println("Apple exists with value:", value)

} else {

fmt.Println("Apple does not exist")

}

* + This method return two values 1st one is value of that key & 2nd is Boolean value (true/false).
* Deleting Elements :
  + You can remove a key-value pair from a map using the delete function.

delete(m, "apple") // Removes the "apple" key and its value from the map

* Iterating Over a Map :
  + You can iterate over all key-value pairs in a map using a for range loop. Note that the order of iteration is random.

for key, value := range m {

fmt.Printf("Key: %s, Value: %d\n", key, value)

}

* Length of a Map:
  + len function to get the number of key-value pairs in the map.

fmt.Println(len(m)) // Output: 2

* Checking if a Map is Nil:

var m map[string]int

if m == nil { fmt.Println("Map is nil") }

**Questions**

 **Map Declaration:**

* How do you declare a map in Go?
* What is the syntax for declaring a map that maps strings to integers?

 **Map Initialization:**

* How do you initialize a map with values in Go?
* Provide an example of initializing a map with some key-value pairs.

 **Accessing Map Elements:**

* How do you access the value associated with a key in a map?
* What happens if you try to access a key that doesn’t exist in the map?

 **Adding and Updating Elements:**

* How do you add a new key-value pair to a map in Go?
* How do you update the value associated with a key in a map?

 **Deleting Map Elements:**

* How do you delete a key-value pair from a map?
* What happens if you try to delete a key that doesn’t exist in the map?

 **Check if Key Exists:**

* How do you check if a key exists in a map in Go?
* Provide an example of checking the existence of a key before accessing its value.

 **Iterating Over a Map:**

* How do you iterate over all key-value pairs in a map using a for loop?
* Provide an example of iterating over a map and printing all keys and values.

 **Map Length:**

* How do you find the number of key-value pairs in a map?
* Write a short code snippet to print the length of a map.
  1. Structure
* In go lang struct is short for structure.
* It is a user define & composite data type.
* It is a collection of variables or members under single name.
* It is used to create complex data structures.
* Following is the basic of structure :
* Defining a Struct :
  + You define a struct using the **type** keyword followed by the struct name and the fields inside curly braces.

type Person struct {

Name string

Age int

Email string

}

* + Variable’s first letter must be capital because of exportation. It can’t accessible outside the block.
* Creating an Instance of a Struct :
  + You can create an instance of a struct by specifying values for its fields.

var p1 Person

p1.Name = "Alice"

p1.Age = 30

p1.Email = "alice@example.com"

// Creating and initializing in one step

p2 := Person{

Name: "Bob",

Age: 25,

Email: "bob@example.com",

}

// Creating an anonymous struct instance

p3 := struct {

Name string

Age int

}{

Name: "Charlie",

Age: 35,

}

* Accessing and Modifying Struct Fields :
  + You can access and modify the fields of a struct using the dot (.) notation.

fmt.Println(p2.Name) // Output: Bob

p2.Age = 26

fmt.Println(p2.Age) // Output: 26

* Anonymous Fields :
  + A struct can have fields without a name, where the type of the field is used as the field name.

type Book struct {

string // anonymous field

int // anonymous field

}

b := Book{"Go Programming", 2024}

fmt.Println(b.string) // Output: Go Programming

fmt.Println(b.int) // Output: 2024

* Structs with Embedded Fields :
  + You can embed one struct within another to create a hierarchical relationship. This is often used for composition.

type Address struct {

City string

State string

}

type Employee struct {

Name string

Age int

Address // embedded struct

}

emp := Employee{

Name: "John",

Age: 28,

Address: Address{

City: "New York",

State: "NY",

},

}

fmt.Println(emp.City) // Output: New York (accessing embedded field directly)

* Zero Value of Structs :
  + When a struct is declared but not explicitly initialized, its fields are set to their zero values (e.g., "" for strings, 0 for integers).

var p Person // p is a zero-valued instance of Person

fmt.Println(p.Name) // Output: "" (empty string)

fmt.Println(p.Age) // Output: 0

* Anonymous Structs :
  + You can define and instantiate an anonymous struct on the fly.

p := struct {

Name string

Age int

}{

Name: "Diana",

Age: 40,

}

fmt.Println(p.Name) // Output: Diana

* Comparing Structs :
  + You can compare structs using the == operator if all fields are comparable.

p1 := Person{Name: "Alice", Age: 30}

p2 := Person{Name: "Alice", Age: 30}

fmt.Println(p1 == p2) // Output: true

* Struct Literals :
  + You can create instances of a struct using literals, either by specifying field names or by using positional values.

// Using field names

p := Person{

Name: "Alice",

Age: 30,

Email: "alice@example.com",

}

// Using positional values (order matters)

p := Person{"Alice", 30, "alice@example.com"}

**Questions**

 **Struct Declaration:**

* How do you declare a struct in Go?
* Provide an example of declaring a struct with fields Name (string) and Age (int).

 **Creating and Initializing Structs:**

* How do you create an instance of a struct in Go?
* What are the different ways to initialize a struct in Go?

 **Accessing Struct Fields:**

* How do you access and modify the fields of a struct in Go?
* Provide an example of setting the Name field of a struct.

 **Anonymous Fields:**

* What are anonymous fields in a struct?
* Provide an example of a struct with an anonymous field.

 **Embedding Structs:**

* How do you embed one struct inside another in Go?
* Explain the concept of "promotion" in struct embedding.

 **Comparing Structs:**

* How can you compare two structs in Go?
* Can you compare all structs directly in Go? Why or why not?

 **Nested Structs:**

* How do you create and work with nested structs in Go?
* Provide an example of accessing a field in a nested struct.

 **Zero Values and Default Initialization:**

* What is the zero value of a struct field in Go?
* How are structs initialized by default when only partial values are provided?

2.0 Pointers

* Pointers are used to stored the memory address of another variable.
* Pointers provide a way to work with memory directly, which can be useful for various programming tasks, including efficient memory management & sharing data between functions.
* Declaring a Pointer :
  + A pointer is declared by prefixing the type with an asterisk (\*). The pointer will store the memory address of a value of that type.

var p \*int // p is a pointer to an int

var q \*float64 // q is a pointer to a float64

* Getting the Address of a Variable :
  + The address of a variable can be obtained using the & operator, which returns the memory address of the variable.

var x int = 10

var p \*int = &x // p holds the address of x

fmt.Println(p) // Output: memory address (e.g., 0xc0000120a8)

* Dereferencing a Pointer :
  + Dereferencing a pointer means accessing the value stored at the memory address the pointer is holding. This is done using the \* operator.

// Modifying the value through the pointer

\*p = 20

fmt.Println(x) // Output: 20 (x is now 20)

* Pointers and Function Arguments :
  + When a function argument is a pointer, changes to the argument within the function affect the original variable.

func increment(val \*int) {

\*val++ // Increment value pointed to by val

}

var x int = 10

increment(&x) // Pass address of x to function

fmt.Println(x) // Output: 11 (x is ncremented)

* Nil Pointers:
  + A pointer that does not point to any memory address is called a nil pointer. The zero value of a pointer is nil.

var p \*int fmt.Println(p) // Output: <nil>

* Pointer Arithmetic (Not Allowed) :
  + Unlike languages like C or C++, Go does not support pointer arithmetic .
  + (e.g., you cannot increment or decrement a pointer to move to the next memory address).
* Pointers to Pointers :
  + Go supports pointers to pointers, allowing you to reference the address of a pointer variable.

var x int = 10

var p \*int = &x

var pp \*\*int = &p // pp is a pointer to p

fmt.Println(\*\*pp) // Output: 10 (dereferencing twice)

* New Function :
  + The new function allocates memory for a variable and returns a pointer to that memory. The allocated memory is zeroed.

p := new(int) // p is a pointer to an int

fmt.Println(\*p) // Output: 0 (zero value of int)

\*p = 100

fmt.Println(\*p) // Output: 100

**Questions**

 **Pointer Basics:**

* What is a pointer in Go?
* How do you declare a pointer to an integer in Go?

 **Address-of Operator (&):**

* What does the & operator do in Go?
* Provide an example of using the & operator to get the address of a variable.

 **Dereferencing Operator (\*):**

* What does the \* operator do when used with a pointer in Go?
* Provide an example of using the \* operator to access the value stored at a pointer.

 **Nil Pointers:**

* What is a nil pointer in Go?
* How do you check if a pointer is nil?

 **Pointer to a Struct:**

* How do you declare a pointer to a struct in Go?
* Provide an example of accessing struct fields through a pointer.

 **Pointer to a Pointer:**

* Can you have a pointer to a pointer in Go? How do you declare it?
* Provide an example of a pointer to a pointer and how to dereference it.

 **Passing Pointers to Functions:**

* How do you pass a pointer to a function in Go?
* What are the advantages of passing a pointer to a function rather than a value?

 **Returning Pointers from Functions:**

* How do you return a pointer from a function in Go?
* Provide an example of a function that returns a pointer to an integer.

 **Pointer Arithmetic:**

* Does Go support pointer arithmetic? Why or why not?
* What are the alternatives in Go if you need to manipulate memory addresses?

2.1 Data Conversion

* Basic Type Conversion :
  + To convert between basic types like integers, floats, and strings, you use a simple type conversion syntax.

var i int = 42

var f float64=float64(i) //convert int to float64

var u uint = uint(i) // converting int to uint

fmt.Println(f) // Output: 42.0

fmt.Println(u) // Output: 42

* String to Integer :
  + To convert a string to an integer, you can use the **strconv** package.

var s string = "123"

i, err := strconv.Atoi(s) // Atoi converts

fmt.Println(i) // Output: 123

* Integer to String :
  + To convert an integer to a string, you can use the **strconv** package or the **fmt.Sprintf** function.

var i int = 123

s := strconv.Itoa(i)

fmt.Println(s) // Output: "123"

s = fmt.Sprintf("%d", i) // Using fmt.Sprintf fmt.Println(s) // Output: "123"

* String to Float :
  + To convert a string to a floating-point number, you use the **strconv.ParseFloat** function.

var s string = "3.14" f, err := strconv.ParseFloat(s, 64) // 64 specifies the precision (float64)

* Float to String :
  + To convert a float to a string, you can use **strconv.FormatFloat** or **fmt.Sprintf**.

var f float64 = 3.14

s := strconv.FormatFloat(f, 'f', 2, 64)

// 'f' format, 2 decimal places, float64 precision

s = fmt.Sprintf("%.2f", f) // Using fmt.Sprintf for formatting

* String to Boolean :
  + To convert a string to a boolean, use **strconv.ParseBool**.

var s string = "true"

b, err := strconv.ParseBool(s)

* Boolean to String :
  + To convert a boolean to a string, use **strconv.FormatBool** or **fmt.Sprintf**.

var b bool = true

s := strconv.FormatBool(b)

s = fmt.Sprintf("%t", b) // Using fmt.Sprintf

* String to Byte Slice :
  + A string can be converted to a byte slice, which is useful when dealing with raw data or binary formats.

var s string = "hello"

b := []byte(s) // converting string to byte slice

fmt.Println(b) // Output: [104 101 108 108]

* Byte Slice to String :

b := []byte{104, 101, 108, 108, 111}

s := string(b) // converting byte slice to string

fmt.Println(s) // Output: "hello"

2.2 String Package

* Basic String Functions :
  + contains :

fmt.Println(strings.contains("Hello, world", "world")) // Output: true

* + containsAny :

fmt.Println(strings.containsAny("Hello, world", "oe")) // Output: true

* + containsRune :

fmt.Println(strings.containsRune("Hello", 'H')) // Output: true

* + count :

fmt.Println(strings.count("banana", "a"))

// Output: 3

* String Modification Functions :
  + Join :

parts := []string{"Go", "is", "fun"}

fmt.Println(strings.Join(parts, " ")) // Output: "Go is fun"

* + Replace :

fmt.Println(strings.Replace("hello", "l", "x", 2)) // Output: "hexxo"

* + Repeat :

fmt.Println(strings.Repeat("Go", 3)) // Output: "GoGoGo"

* + Split :

fmt.Println(strings.Split("a,b,c", ","))

// Output: ["a" "b" "c"]

* + ToLower :

fmt.Println(strings.ToLower("HELLO"))

// Output: "hello"

* + ToUpper :

fmt.Println(strings.ToUpper("hello"))

// Output: "HELLO"

* + Trim :

fmt.Println(strings.Trim("!!!Hello, Gophers!!!", "!")) // Output: "Hello, Gophers"

* String Comparison Functions :
  + compare :

fmt.Println(strings.compare("a", "b"))

// Output: -1

fmt.Println(strings. compare ("b", "a"))

// Output: 1

fmt.Println(strings. compare ("a", "a"))

// Output: 0

* + EqualFold :
    - Compares two strings case-insensitively.

fmt.Println(strings.EqualFold("Go", "go"))

// Output: true

2.3 Time Package

* The time package in Go provides functionality for measuring and displaying time.
* It includes features for working with dates, times.
* Time Package has fixed reference date & time **“2006-01-02 15:04:05 PM”**, MST (Mountain Standard Time)for formatting.
* This format is chosen because it is the time when Go was officially announced.
* Lets break down **“2006-01-02 15:04:05 PM” :**
  + **2006 :** Year Part.
  + **01 :** Month Part.
  + **02 :** Date Part.
  + **15 :** Hour Part of time in 24-hour format.
  + **04 :** Minutes Part.
  + **05 :** Second Part.
* If we want in 12-hour format then **15** replaced with **3**. And for day formatting it is **Monday**.
* Format(“2006-01-02 15:04:05”) tells Go to replace the year, month, date, hour, minute, second as per the current time.
* time.Now:
  + Retrieves the current local time.

now := time.Now()

fmt.Println(now) // Output: 2024-08-13 12:34:56.789 +0200 CEST m=+0.000123456

* Time.Format :
  + Formats a **time.Time** object into a string according to a specific layout.

now := time.Now()

formatted := now.Format("2006-01-02 15:04:05")

fmt.Println(formatted) // Output: "2024-08-30 12:34:56"

* time.Parse :
  + Converts a string representation of time into a time.Time object according to a specific layout.

layout := "2006-01-02 15:04:05"

str := "2024-08-13 12:34:56"

t, \_ := time.Parse(layout, str)

fmt.Println(t) // Output: 2024-08-13 12:34:56

2.4 Defer Keyword

* The **defer** keyword in Go is used to call the function just before the surrounded function exits or return.
* The simplest way to use defer is by placing it before a function call.
  + defer fmt.Println("Deferred execution")
* It's commonly used for resource management, such as closing files.
* If multiple defer statements are used in a single function, they are executed in **last-in, first-out (LIFO)** order.
* Basic Usage of defer :

func main() {

fmt.Println("Start")

defer fmt.Println("Deferred execution") fmt.Println("End")

}

* Output :

Start

End

Deferred execution

* Multiple defer Statements :

func main() {

defer fmt.Println("First defer")

defer fmt.Println("Second defer")

defer fmt.Println("Third defer") fmt.Println("Function body")

}

* + Output :

Function body

Third defer

Second defer

First defer

* Using defer for Resource Management :

func main() {

file, err := os.Open("example.txt")

if err != nil {

fmt.Println("Error opening file:", err)

return

}

defer file.Close() // Ensures the file is closed when the function ends

fmt.Println("File opened successfully")

}

2.5 File Handling

* File management in Go includes working with files on the file system, including tasks like creating, opening, reading, writing, and deleting files.
* The **os** package in Go provides a wide range of functions to perform these file operations.
* **Opening and Creating Files** :
  + **os.Open** : Opens an existing file for reading. Returns an **\*os.File** and an error.
  + **os.Create** : Creates a new file or truncates an existing one. Returns an **\*os.File** and an error.

func main() {

// Opening a file

file, err := os.Open("example.txt")

if err != nil {

fmt.Println("Error opening file:", err)

return

}

defer file.Close() // Ensure the file is closed

// Creating a new file

newFile, err := os.Create("newfile.txt")

if err != nil {

fmt.Println("Error creating file:", err)

return

}

defer newFile.Close() // Ensure the file is closed }

* **Closing Files** :
  + **File.Close**: Closes an open file. It’s important to close files to free up resources.

defer newFile.Close()

* **Reading Files** :
  + There are two ways to read file.
  + **File.Read** : Reads up to len(b) bytes from the file into b.
  + **io/ioutil.ReadFile**: Reads the entire content of a file into memory.
  + **io/ioutil.ReadFile** is easy to use but it can’t useful in large files.

func main() {

// Reading file content with os.File.Read

file, err := os.Open("example.txt")

if err != nil {

fmt.Println("Error opening file:", err)

return

}

defer file.Close()

buffer := make([]byte, 100)

for {

bytesRead, err := file.Read(buffer)

if err==io.EOF{

break

}

if err != nil {

fmt.Println("Error reading file:", err)

return

}

fmt.Println(string(buffer[:bytesRead]))

}

// Reading entire file content with ioutil.ReadFile content, err := ioutil.ReadFile("example.txt")

if err != nil {

fmt.Println("Error reading file:", err)

return

}

fmt.Println(string(content))

}

* **Writing to Files** :
  + **File.Write**: Writes data to the file.
  + **File.WriteString**: Writes a string to the file.
  + **io/ioutil.WriteFile**: Writes data to a file, creating or truncating it.

func main() {

// Writing to a file

file, err := os.Create("newfile.txt")

if err != nil {

fmt.Println("Error creating file:", err)

return

}

defer file.Close()

stringWritten, err := file.WriteString("Hello and welcome to GoLang with us!!!")

if err != nil {

fmt.Println("Error writing string to file:", err) return

}

fmt.Printf("Wrote %d bytes to file\n", stringWritten)

}

* **Deleting Files** :
  + **os.Remove**: Deletes a file or directory.
  + **os.RemoveAll**: Deletes a directory and its contents recursively.

err := os.Remove("newfile.txt")

* Renaming Files :
  + **os.Rename**: Renames a file or directory.

err := os.Rename("newfile.txt", "renamedfile.txt")