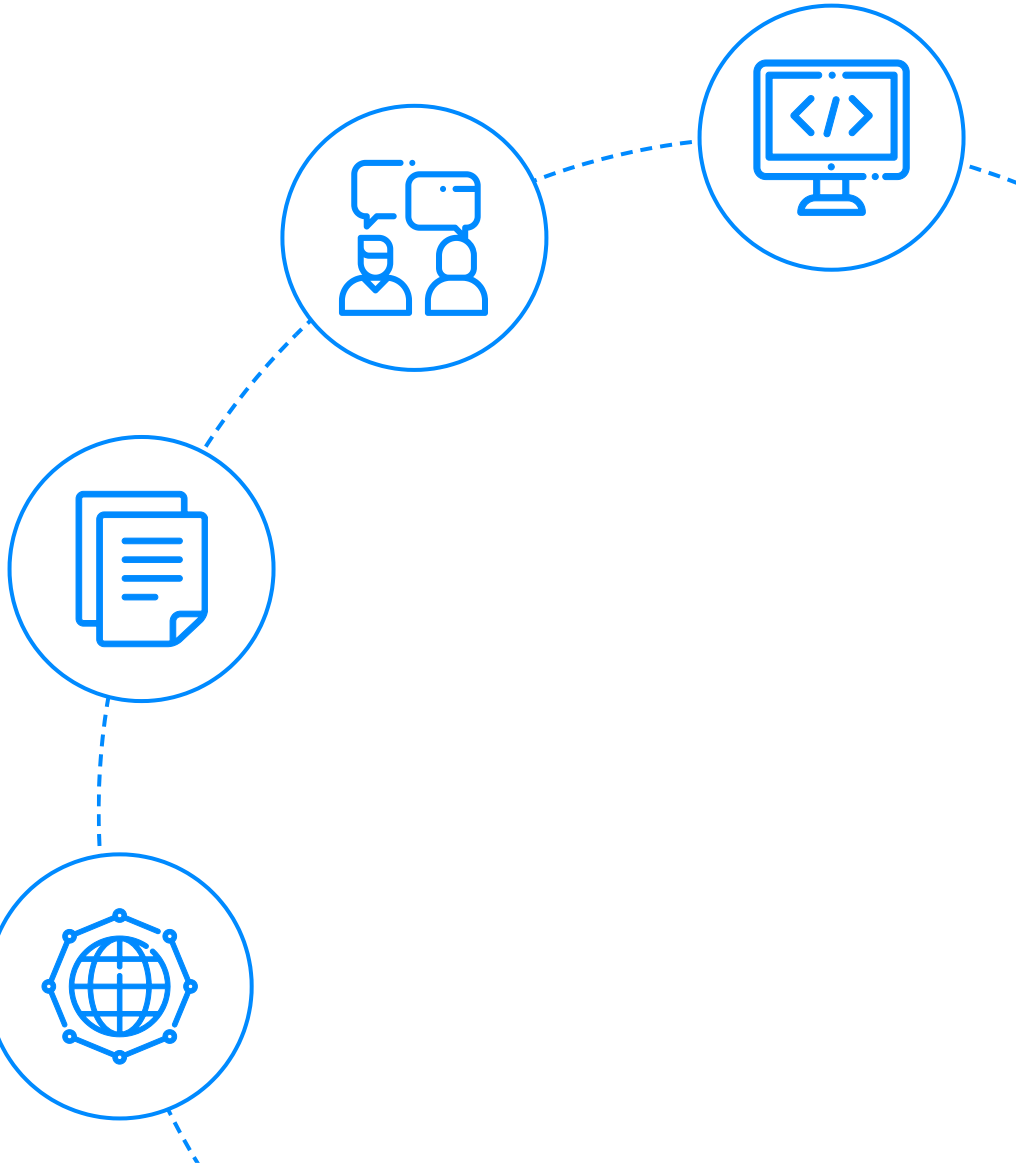
InterviewBit

Golang Interview Questions



To view the live version of the page, click here.

Contents

Golang Interview Questions for Freshers

Golang Interview Questions for Experienced

Golang Interview Questions for Experienced

(.....Continued)

19. What do you understand by Type Assertion in Go?
20. How will you check the type of a variable at runtime in Go?
21. Is the usage of Global Variables in programs implementing goroutines recommended?
22. What are the uses of an empty struct?
23. How can we copy a slice and a map in Go?
24. How is GoPATH different from GoROOT variables in Go?
25. In Go, are there any good error handling practices?
26. Which is safer for concurrent data access? Channels or Maps?
27. How can you sort a slice of custom structs with the help of an example?
28. What do you understand by Shadowing in Go?
29. What do you understand by variadic functions in Go?
30. What do you understand by byte and rune data types? How are they represented?

Golang Programs

31. Write a Go program to swap variables in a list?
32. Write a GO Program to find factorial of a given number.
33. Write a Go program to find the nth Fibonacci number.
34. Write a Golang code for checking if the given characters are present in a string.
35. Write a Go code to compare two slices of a byte.

Let's get Started

Golang or most popularly known as Go is one of the youngest programming languages that was released in the year 2012 at Google by developers Robert Griesemer, Rob Pike and Ken Thompson. It is said that Golang was born out of frustration with the demerits of the existing programming languages.

Go is a high level, open-source programming language that was mainly developed keeping in mind the efficiency of code without compromising on the simplicity and faster compilation time to help develop software applications at a faster pace. Companies like Google, Apple, Uber are using Golang due to its proven ability of less learning time, faster code development, improved runtime efficiency, reduced bugs, concurrency, garbage collection strategies and so on.

In this article, we will see the most commonly asked interview questions for both freshers and experienced in Golang.

Golang Interview Questions for Freshers

1. What is Golang?

- Go is a high level, general-purpose programming language that is very strongly and statically typed by providing support for garbage collection and concurrent programming.
- In Go, the programs are built by using packages that help in managing the dependencies efficiently. It also uses a compile-link model for generating executable binaries from the source code. Go is a simple language with elegant and easy to understand syntax structures. It has a built-in collection of powerful standard libraries that helps developers in solving problems without the need for third party packages. Go has first-class support for Concurrency having the ability to use multi-core processor architectures to the advantage of the developer and utilize memory efficiently. This helps the applications scale in a simpler way.

2. Why should one learn Golang? What are the advantages of Golang over other languages?

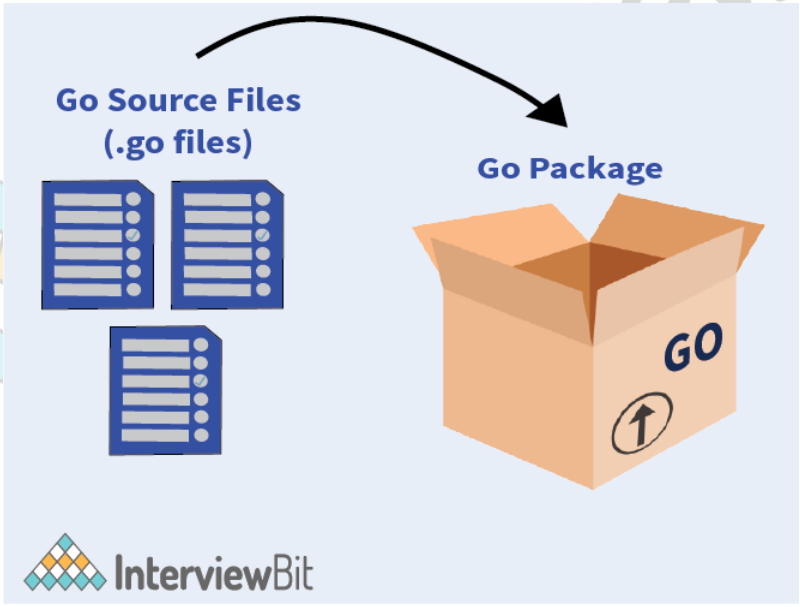
Go language follows the principle of maximum effect with minimum efforts. Every feature and syntax of Go was developed to ease the life of programmers. Following are the advantages of Go Language:

- **Simple and Understandable:** Go is very simple to learn and understand. There are no unnecessary features included. Every single line of the Go code is very easily readable and thereby easily understandable irrespective of the size of the codebase. Go was developed by keeping simplicity, maintainability and readability in mind.
- **Standard Powerful Library:** Go supports all standard libraries and packages that help in writing code easily and efficiently.
- **Support for concurrency:** Go provides very good support for concurrency using Go Routines or channels. They take advantage of efficient memory management strategies and multi-core processor architecture for implementing concurrency.
- **Static Type Checking:** Go is a very strong and statically typed programming language. Statically typed means every variable has types assigned to it. The data type cannot be changed once created and strongly typed means that there are rules and restrictions while performing type conversion. This ensures that the code is type-safe and all type conversions are handled efficiently. This is done for reducing the chances of errors at runtime.
- **Easy to install Binaries:** Go provides support for generating binaries for the applications with all required dependencies. These binaries help to install tools or applications written in Go very easily without the need for a Go compiler or package managers or runtimes.
- **Good Testing Support:** Go has good support for writing unit test cases along with our code. There are libraries that support checking code coverage and generating code documentation.

3. What are Golang packages?

Go Packages (in short `pkg`) are nothing but directories in the Go workspace that contains Go source files or other Go packages themselves. Every single piece of code starting from variables to functions are written in the source files are in turn stored in a linked package. Every source file should belong to a package.

From the image below, we can see that a Go Package is represented as a box where we can store multiple Go source files of the .go extension. We can also store Go packages as well within a package.



The package is declared at the top of the Go source file as `package <package_name>`

The packages can be imported to our source file by writing: `import <package_name>`

An example of the Go package is `fmt`. This is a standard Go Package that has formatting and printing functionalities such as `Println()`.

4. Is Golang case sensitive or insensitive?

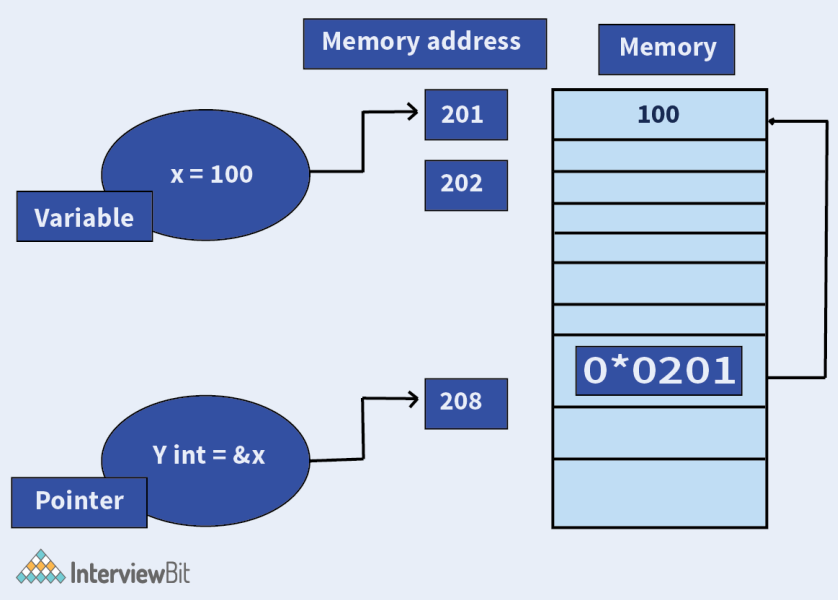
Go is a case-sensitive language.

5. What are Golang pointers?

Go Pointers are those variables that hold the address of any variables. Due to this, they are called special variables. Pointers support two operators:

- `*` **operator:** This operator is called a dereferencing operator and is used for accessing the value in the address stored by the pointer.
- `&` **operator:** This operator is called the address operator and is used for returning the address of the variable stored in the pointer.

This is illustrated in the diagram below. Here, consider we have a variable x assigned to 100. We store x in the memory address 0x0201. Now, when we create a pointer of the name Y for the variable x, we assign the value as `&x` for storing the address of variable x. The pointer variable is stored in address 0x0208. Now to get the value stored in the address that is stored in the pointer, we can just write `int z:= *Y`



Pointers are used for the following purposes:

- Allowing function to directly mutate value passed to it. That is achieving pass by reference functionality.
- For increasing the performance in the edge cases in the presence of a large data structure. Using pointers help to copy large data efficiently.
- Helps in signifying the lack of values. For instance, while unmarshalling JSON data into a struct, it is useful to know if the key is present or absent then the key is present with 0 value.

6. What do you understand by Golang string literals?

String literals are those variables storing string constants that can be a single character or that can be obtained as a result of the concatenation of a sequence of characters. Go provides two types of string literals. They are:

- **Raw string literals:** Here, the values are uninterrupted character sequences between backquotes. For example:

```
`interviewbit`
```

- **Interpreted string literals**: Here, the character sequences are enclosed in double quotes. The value may or may not have new lines. For example:

```
"Interviewbit
Website"
```

7. What is the syntax used for the for loop in Golang? Explain.

Go language follows the below syntax for implementing for loop.

```
for [condition |  ( init; condition; increment ) | Range]
{
   statement(s);
   //more statements
}
```

The for loop works as follows:

- The `init` steps gets executed first. This is executed only once at the beginning of the loop. This is done for declaring and initializing the loop control variables. This field is optional as long as we have initialized the loop control variables before. If we are not doing anything here, the semicolon needs to be present.
- The `condition` is then evaluated. If the `condition` is satisfied, the loop body is executed.
    - If the condition is not satisfied, the control flow goes to the next statement after the for loop.
    - If the condition is satisfied and the loop body is executed, then the control goes back to the `increment` statement which updated the loop control variables. The condition is evaluated again and the process repeats until the condition becomes false.
- If the `Range` is mentioned, then the loop is executed for each item in that Range.

Consider an example for `for` loop. The following code prints numbers from 1 to 5.

```
package main

import "fmt"

func main() {
    // For loop to print numbers from 1 to 5
    for j := 1; j <= 5; j++ {
        fmt.Println(j)
    }

}
```

The output of this code is:

```
1
2
3
4
5
```

8. What do you understand by the scope of variables in Go?

The variable scope is defined as the part of the program where the variable can be accessed. Every variable is statically scoped (meaning a variable scope can be identified at compile time) in Go which means that the scope is declared at the time of compilation itself. There are two scopes in Go, they are:

- **Local variables** - These are declared inside a function or a block and is accessible only within these entities.
- **Global variables** - These are declared outside function or block and is accessible by the whole source file.

9. What do you understand by goroutine in Golang?

A goroutine is nothing but a function in Golang that usually runs concurrently or parallelly with other functions. They can be imagined as a lightweight thread that has independent execution and can run concurrently with other routines. Goroutines are entirely managed by Go Runtime. Goroutines help Golang achieve concurrency.

- In Golang, the main function of the main package is considered the main goroutine. It is the starting point of all other goroutines. These goroutines have the power to start their goroutines. Once the execution of the main goroutine is complete, it means that the program has been completed.
- We can start a goroutine by just specifying the `go` keyword before the method call. The method will now be called and run as a goroutine. Consider an example below:

```
package main
import (
    "fmt"
    "time"
)
func main() {
    go sampleRoutine()
    fmt.Println("Started Main")
    time.Sleep(1 * time.Second)
    fmt.Println("Finished Main")
}

func sampleRoutine() {
    fmt.Println("Inside Sample Goroutine")
}
```

In this code, we see that the sampleRoutine() function is called by specifying the keyword go before it. When a function is called a goroutine, the call will be returned immediately to the next line of the program statement which is why “Started Main” would be printed first and the goroutine will be scheduled and run concurrently in the background. The sleep statement ensures that the goroutine is scheduled before the completion of the main goroutine. The output of this code would be:

```
Started Main
Inside Sample Goroutine
Finished Main
```

10. Is it possible to return multiple values from a function in Go?

Yes. Multiple values can be returned in Golang by sending comma-separated values with the return statement and by assigning it to multiple variables in a single statement as shown in the example below:

```
package main
import (
    "fmt"
)

func reverseValues(a,b string)(string, string){
    return b,a    //notice how multiple values are returned
}

func main(){
    val1,val2:= reverseValues("interview","bit")    // notice how multiple values are a
    fmt.Println(val1, val2)
}
```

In the above example, we have a function reverseValues which simply returns the inputs in reverse order. In the main goroutine, we call the reverseValues function and the values are assigned to values val1 and val2 in one statement. The output of the code would be

```
bit interview
```

11. Is it possible to declare variables of different types in a single line of code in Golang?

Yes, this can be achieved by writing as shown below:

```
var a,b,c= 9, 7.1, "interviewbit"
```

Here, we are assigning values of a type Integer number, Floating-Point number and string to the three variables in a single line of code.

12. What is “slice” in Go?

Slice in Go is a lightweight data structure of variable length sequence for storing homogeneous data. It is more convenient, powerful and flexible than an array in Go. Slice has 3 components:

- **Pointer:** This is used for pointing to the first element of the array accessible via slice. The element doesn't need to be the first element of the array.
- **Length:** This is used for representing the total elements count present in the slice.
- **Capacity:** This represents the capacity up to which the slice can expand.

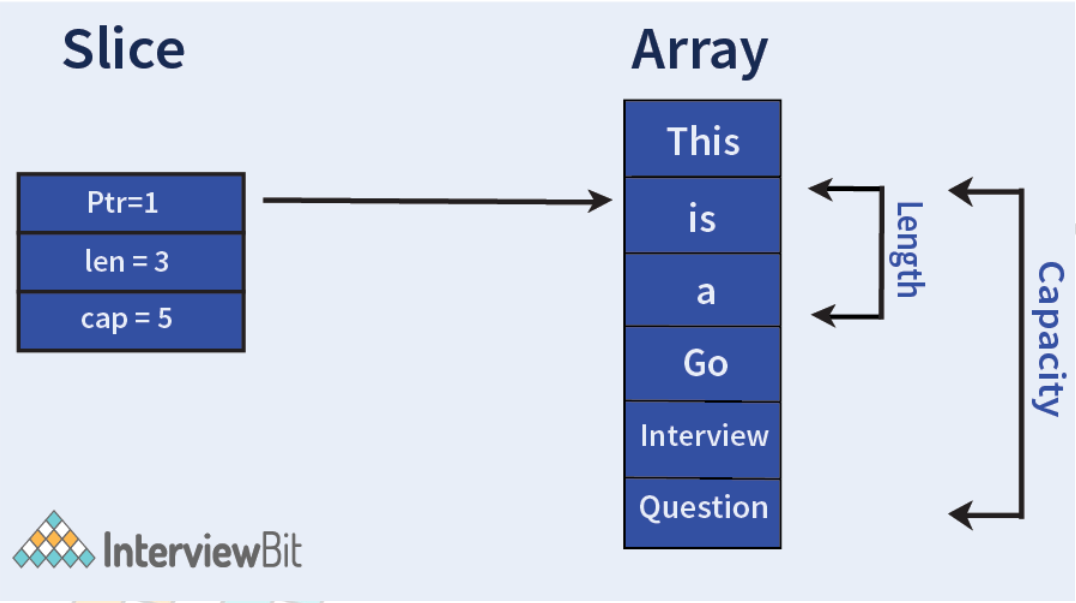
For example: Consider an array of name arr having the values "This","is", "a","Go","interview","question".

```
package main

import "fmt"

func main() {

    // Creating an array
    arr := [6]string{"This","is", "a","Go","interview","question"}

    // Print array
    fmt.Println("Original Array:", arr)

    // Create a slice
    slicedArr := arr[1:4]

    // Display slice
    fmt.Println("Sliced Array:", slicedArr)

    // Length of slice calculated using len()
    fmt.Println("Length of the slice: %d", len(slicedArr))

    // Capacity of slice calculated using cap()
    fmt.Println("Capacity of the slice: %d", cap(slicedArr))
}
```

Here, we are trying to slice the array to get only the first 3 words starting from the word at the first index from the original array. Then we are finding the length of the slice and the capacity of the slice. The output of the above code would be:

```
Original Array: [This is a Go interview question ]
Sliced Array: [is a Go]
Length of the slice: 3
The capacity of the slice: 5
```

The same is illustrated in the diagram below:



13. What are Go Interfaces?

Go interfaces are those that have a defined set of method signatures. It is a custom type who can take values that has these methods implementation. The interfaces are abstract which is why we cannot create its instance. But we can create a variable of type interface and that variable can then be assigned to a concrete value that has methods required by the interface. Due to these reasons, an interface can act as two things:

- Collection of method signatures
- Custom types

They are created by using the `type` keyword followed by the name needed for the interface and finally followed by the keyword `interface`. The syntax goes as follows:

```
type name_of_interface interface{

// Method signatures

}
```

Consider an example of creating an interface of the name “golangInterfaceDemo” having two methods demo_func1() and demo_func2(). The interface will be defined as:

```
// Create an interface
type golangInterfaceDemo interface{
    // Methods
    demo_func1() int
    demo_func2() float64
}
```

Interface also promotes abstraction. In Golang, we can use interfaces for creating common abstractions which can be used by multiple types by defining method declarations that are compatible with the interface. Conside the following example:

```
package main

import "fmt"

// "Triangle" data type
type Triangle struct {
    base, height float32
}

// "Square" data type
type Square struct {
    length float32
}

// "Rectangle" data type
type Rectangle struct {
    length, breadth float32
}

// To calculate area of triangle
func (triangle Triangle) Area() float32 {
    return 0.5 * triangle.base * triangle.height
}

// To calculate area of square
func (square Square) Area() float32 {
    return square.length * square.length
}

// To calculate area of rectangle
func (rect Rectangle) Area() float32 {
    return rect.length * rect.breadth
}

// Area interface for achieving abstraction
type Area interface {
    Area() float32
}

func main() {
    // Declare and assign values to varaibles
    triangleObject := Triangle{base: 20, height: 10}
    squareobject := Square{length: 25}
    rectObject := Rectangle{length: 15, breadth: 20}

    // Define a variable of type interface
    var shapeObject Area

    // Assign to "Triangle" type variable to the Area interface
    shapeObject = triangleObject
    fmt.Println("Triangle Area = ", shapeObject.Area())

    // Assign to "Square" type variable to the Area interface
    shapeObject = squareobject
    fmt.Println("Square Area = ", shapeObject.Area())

    // Assign to "Rectangle" type variable to the Area interface
```

In the above example, we have created 3 types for the shapes triangle, square and rectangle. We have also defined 3 Area() functions that calculate the area of the shapes based on the input object type passed. We have also defined an interface named Area and we have defined the method signature Area() within it. In the main function, we are creating the objects, assigning each object to the interface and calculating the area by calling the method declared in the interface. Here, we need not know specifically about the function that needs to be called. The interface method will take care of this considering the object type. This is called abstraction. The output of the above code will be:

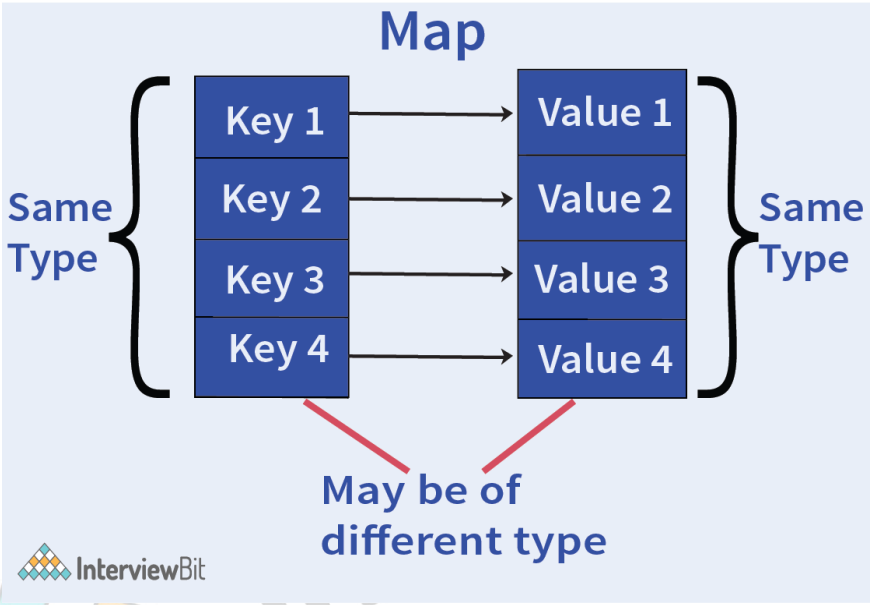
```
Triangle Area =  100
Square Area =  625
Rectangle Area =  300
```

14. Why is Golang fast compared to other languages?

Golang is faster than other programming languages because of its simple and efficient memory management and concurrency model. The compilation process to machine code is very fast and efficient. Additionally, the dependencies are linked to a single binary file thereby putting off dependencies on servers.

15. How can we check if the Go map contains a key?

A map, in general, is a collection of elements grouped in key-value pairs. One key refers to one value. Maps provide faster access in terms of O(1) complexity to the values if the key is known. A map is visualized as shown in the image below:



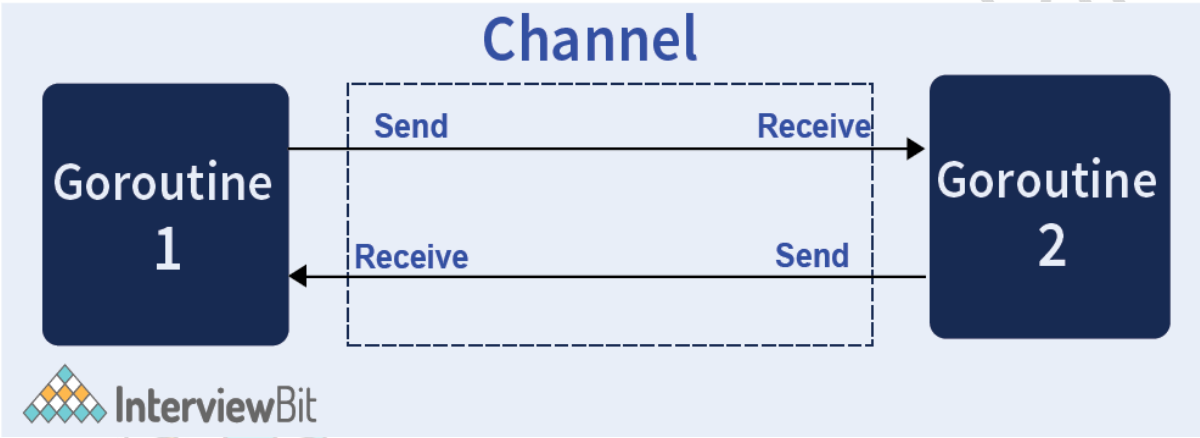
Once the values are stored in key-value pairs in the map, we can retrieve the object by using the key as `map_name[key_name]` and we can check if the key, say “foo”, is present or not and then perform some operations by using the below code:

```
if val, isExists := map_obj["foo"]; isExists {
    //do steps needed here
}
```

From the above code, we can see that two variables are being initialized. The val variable would get the value corresponding to the key “foo” from the map. If no value is present, we get “zero value” and the other variable isExists will get a bool value that will be set to true if the key “foo” is present in the map else false. Then the isExists condition is evaluated, if the value is true, then the body of the if would be executed.

16. What are Go channels and how are channels used in Golang?

Go channel is a medium using which goroutines communicate data values with each other. It is a technique that allows data transfer to other goroutines. A channel can transfer data of the same type. The data transfer in the channel is bidirectional meaning the goroutines can use the same channel for sending or receiving the data as shown in the image below:



A channel can be created by adding the `chan` keyword as shown in the syntax below:

```
var channel_name chan Type
```

It can also be created by using the make() function as:

```
channel_name:= make(chan Type)
```

To send the data to a channel, we can use the `<-` operator as shown in the syntax:

```
channel_name <- element
```

To receive data sent by the send operator, we can use the below syntax:

```
element := <-Mychannel
```

Golang Interview Questions for Experienced

17. What do you understand by each of the functions demo_func() as shown in the below code?

```
//DemoStruct definition
type DemoStruct struct {
    Val int
}
//A.
func demo_func() DemoStruct {
    return DemoStruct{Val: 1}
}
//B.
func demo_func() *DemoStruct {
    return &DemoStruct{}
}
//C.
func demo_func(s *DemoStruct) {
    s.Val = 1
}
```

A. Since the function has a return type of the struct, the function returns a copy of the struct by setting the value as 1.
B. Since the function returns `*DemoStruct`, which is a pointer to the struct, it returns a pointer to the struct value created within the function.
C. Since the function expects the existing struct object as a parameter and in the function, we are setting the value of its attribute, at the end of execution the value of Val variable of the struct object is set to 1.

18. Can you format a string without printing?

Yes, we can do that by using the Sprintf command as shown in the example below:

```
return fmt.Sprintf ("Size: %d MB.", 50)
```

The `fmt.Sprintf` function formats a string and returns the string without printing it.

19. What do you understand by Type Assertion in Go?

The type assertion takes the interface value and retrieves the value of the specified explicit data type. The syntax of Type Assertion is:

```
t := i.(T)
```

Here, the statement asserts that the interface value i has the concrete type T and assigns the value of type T to the variable t. In case i does not have concrete type T, then the statement will result in panic.

For testing, if an interface has the concrete type, we can do it by making use of two values returned by type assertion. One value is the underlying value and the other is a bool value that tells if the assertion is completed or not. The syntax would be:

```
t, isSuccess := i.(T)
```

Here, if the interface value i have T, then the underlying value will be assigned to t and the value of isSuccess becomes true. Else, the isSuccess statement would be false and the value of t would have the zero value corresponding to type T. This ensures there is no panic if the assertion fails.

20. How will you check the type of a variable at runtime in Go?

In Go, we can use a special type of switch for checking the variable type at runtime. This switch statement is called a “type switch”.

Consider the following piece of code where we are checking for the type of variable v and performing some set of operations.

```
switch v := param.(type) {
default:
    fmt.Printf("Unexpected type %T", v)
case uint64:
    fmt.Println("Integer type")
case string:
    fmt.Println("String type")
}
```

In the above code, we are checking for the type of variable v, if the type of variable is uint64, then the code prints “Integer type”. If the type of variable is a string, the code prints “String type”. If the type doesn't match, the default block is executed and it runs the statements in the default block.

21. Is the usage of Global Variables in programs implementing goroutines recommended?

Using global variables in goroutines is not recommended because it can be accessed and modified by multiple goroutines concurrently. This can lead to unexpected and arbitrary results.

22. What are the uses of an empty struct?

Empty struct is used when we want to save memories. This is because they do not consume any memory for the values. The syntax for an empty struct is:

```
a := struct{}{}
```

The size of empty struct would return 0 when using `println(unsafe.Sizeof(a))`

The important use of empty struct is to show the developer that we do not have any value. The purpose is purely informational. Some of the examples where the empty struct is useful are as follows:

- **While implementing a data set:** We can use the empty struct to implement a dataset. Consider an example as shown below.

```
map_obj := make(map[string]struct{})
for _, value := range []string{"interviewbit", "golang", "questions"} {
   map_obj[value] = struct{}{}
}
fmt.Println(map_obj)
```

The output of this code would be:

```
map[interviewbit:{} golang:{} questions:{}]
```

Here, we are initializing the value of a key to an empty struct and initializing the map_obj to an empty struct.

- In graph traversals in the map of tracking visited vertices. For example, consider the below piece of code where we are initializing the value of vertex visited empty struct.

```
visited := make(map[string]struct{})
for _, isExists := visited[v]; !isExists {
    // First time visiting a vertex.
    visited[v] = struct{}{}
}
```

- When a channel needs to send a signal of an event without the need for sending any data. From the below piece of code, we can see that we are sending a signal using sending empty struct to the channel which is sent to the workerRoutine.

```
func workerRoutine(ch chan struct{}) {
    // Receive message from main program.
    <-ch
    println("Signal Received")

    // Send a message to the main program.
    close(ch)
}

func main() {
    //Create channel
    ch := make(chan struct{})

    //define workerRoutine
    go workerRoutine(ch)

    // Send signal to worker goroutine
    ch <- struct{}{}

    // Receive a message from the workerRoutine.
    <-ch
    println(“Signal Received")

}
```

The output of the code would be:

```
Signal Received
Signal Received
```

23. How can we copy a slice and a map in Go?

- **To copy a slice:** We can use the built-in method called `copy()` as shown below:

```
slice1 := []int{1, 2}
slice2 := []int{3, 4}
slice3 := slice1
copy(slice1, slice2)
fmt.Println(slice1, slice2, slice3)
```

In the above example, we are copying the value of `slice2` into `slice1` and we are using the variable `slice3` for holding a reference to the original slice to check if the slice has been copied or not. The output of the above code would be:

```
[3 4] [3 4] [3 4]
```

If we want to copy the slice description alone and not the contents, then we can do it by using the = operator as shown in the code below:

```
slice1 := []int{1, 2}
slice2 := []int{3, 4}
slice3 := slice1
slice1 = slice2
fmt.Println(slice1, slice2, slice3)
```

The output of the code will be:

```
[3 4] [3 4] [1 2]
```

Here, we can see that the contents of slice3 are not changed due to the = operator.

- **To copy a map in Go:** We can copy a map by traversing the keys of the map. There is no built-in method to copy the map. The code for achieving this will be:

```
map1 := map[string]bool{"Interview": true, "Bit": true}
map2 := make(map[string]bool)
for key, value := range map1 {
    map2[key] = value
}
```

From this code, we are iterating the contents of map1 and then adding the values to map2 to the corresponding key.

If we want to copy just the description and not the content of the map, we can again use the = operator as shown below:

```
map1 := map[string]bool{"Interview": true, "Bit": true}
map2 := map[string]bool{"Interview": true, "Questions": true}
map3 := map1
map1 = map2    //copy description
fmt.Println(map1, map2, map3)
```

The output of the below code would be:

```
map[Interview:true Questions:true] map[Interview:true Questions:true] map[Interview:tru
```

24. How is GoPATH different from GoROOT variables in Go?

The GoPATH variable is an environment variable that is used for symbolizing the directories out of `$GOROOT` which combines the source and the binaries of Go Projects. The GoROOT variable determines where the Go SDK is located. We do not have to modify the variable unless we plan to use multiple Go versions. The GoPATH determines the root of the workspace whereas the GoROOT determines the location of Go SDK.

25. In Go, are there any good error handling practices?

In Go, the errors are nothing but an interface type where any type implementing the single Error() method is considered as an error. Go does not have try/catch methods as in other programming languages for handling the errors. They are instead returned as normal values. Following is the syntax for creating the error interface:

```
type error_name interface {
    Error() string
}
```

We use this whenever we apprehend that there are possibilities where a function can go wrong during type conversions or network calls. The function should return an error as its return variable if things go wrong. The caller has to check this error value and identify the error. Any value other than nil is termed as an error.

As part of good error handling practices, guard classes should be used over if-else statements. They should also be wrapped in a meaningful way as they can be passed up in the call stack. Errors of the same types should not be logged or handled multiple times.

26. Which is safer for concurrent data access? Channels or Maps?

Channels are safe for concurrent access because they have blocking/locking mechanisms that do not let goroutines share memory in the presence of multiple threads.

Maps are unsafe because they do not have locking mechanisms. While using maps, we have to use explicit locking mechanisms like mutex for safely sending data through goroutines.

27. How can you sort a slice of custom structs with the help of an example?

We can sort slices of custom structs by using sort.Sort and sort.Stable functions. These methods sort any collection that implements sort.Interface interface that has Len(), Less() and Swap() methods as shown in the code below:

```
type Interface interface {
        // Find number of elements in collection
        Len() int

        // Less method is used for identifying which elements among index i and j are l
        Less(i, j int) bool

        // Swap method is used for swapping elements with indexes i and j
        Swap(i, j int)
}
```

Consider an example of a `Human` Struct having name and age attributes.

```
type Human struct {
    name string
    age  int
}
```

Also, consider we have a slice of struct Human of type AgeFactor that needs to be sorted based on age. The AgeFactor implements the methods of the sort.Interface. Then we can call sort.Sort() method on the audience as shown in the below code:

```
// AgeFactor implements sort.Interface that sorts the slice based on age field.
type AgeFactor []Human
func (a AgeFactor) Len() int           { return len(a) }
func (a AgeFactor) Less(i, j int) bool { return a[i].age < a[j].age }
func (a AgeFactor) Swap(i, j int)      { a[i], a[j] = a[j], a[i] }

func main() {
    audience := []Human{
        {"Alice", 35},
        {"Bob", 45},
        {"James", 25},
    }
    sort.Sort(AgeFactor(audience))
    fmt.Println(audience)
}
```

This code would output:

```
[{James 25} {Alice 35} {Bob 45}]
```

28. What do you understand by Shadowing in Go?

Shadowing is a principle when a variable overrides a variable in a more specific scope. This means that when a variable is declared in an inner scope having the same data type and name in the outer scope, the variable is said to be shadowed. The outer variable is declared before the shadowed variable.

Consider a code snippet as shown below:

```
var numOfCars = 2    // Line 1
type Car struct{
    name string
    model string
    color string
}
cars:= [{
            name:"Toyota",
            model:"Corolla",
            color:"red"
        },
        {
        name:"Toyota",
            model:"Innova",
            color:"gray"
    }]

func countRedCars(){
    for i:=0; i<numOfCars; i++{
    if cars[i].color == "red" {
        numOfCars +=1    // Line 2
        fmt.Println("Inside countRedCars method ", numOfCars)    //Line 3
    }
    }
}
```

Here, we have a function called countRedCars where we will be counting the red cars. We have the numOfCars variable defined at the beginning indicated by the `Line 1` comment. Inside the countRedCars method, we have an if statement that checks whether the colour is red and if red then increments the numOfCars by 1. The interesting point here is that the value of the numCars variable after the end of the if statement will not be affecting the value of the numOfCars variable in the outer scope.

29. What do you understand by variadic functions in Go?

The function that takes a variable number of arguments is called a variadic function. We can pass zero or more parameters in the variadic function. The best example of a variadic function is `fmt.Printf` which requires one fixed argument as the first parameter and it can accept any arguments.

- The syntax for the variadic function isHere, we see that the type of the last parameter is preceded by the ellipsis symbol (`...`) which indicates that the function can take any number of parameters if the type is specified.
- Inside the variadic function, the `...type` can be visualised as a slice. We can also pass the existing slice (or multiple slices) of the mentioned type to the function as a second parameter. When no values are passed in variadic function, the slice is treated as nil.
- These functions are generally used for string formatting.
- Variadic parameter can not be specified as return value, but we can return the variable of type slice from the function.

Consider an example code below:

```
func function_name(arg1, arg2...type)type{
    // Some statements
}
```

```
package main

import(
    "fmt"
    "strings"
)

// Variadic function to join strings and separate them with hyphen
func joinstring(element...string)string{
    return strings.Join(element, "-")
}

func main() {

  // To demonstrate zero argument
   fmt.Println(joinstring())

   // To demonstrate multiple arguments
   fmt.Println(joinstring("Interview", "Bit"))
   fmt.Println(joinstring("Golang", "Interview", "Questions"))

}
```

Here, we have a variadic function called joinstring that takes a variable number of arguments of a type string. We are trying to join the arguments separated by the hyphen symbol. We are demonstrating the variadic function behaviour by first passing 0 arguments and then passing multiple arguments to the function. The output of this code is:

```
Interview-Bit
Golang-Interview-Questions
```

30. What do you understand by byte and rune data types? How are they represented?

`byte` and `rune` are two integer types that are aliases for `uint8` and `int32` types respectively.
The byte represents ASCII characters whereas the rune represents a single Unicode character which is UTF-8 encoded by default.

- The characters or rune literals can be represented by enclosing in single quotes like `'a'` , `'b'` , `'\n'` .
- Rune is also called a Code point and can also be a numeric value. For example, `0x61` in hexadecimal corresponds to the rune literal `a` .

Golang Programs

31. Write a Go program to swap variables in a list?

Consider we have num1=2, num2=3. To swap these two numbers, we can just write:

```
num1,num2 = num2, num1
```

The same logic can be extended to a list of variables as shown below:

```
package main

import "fmt"

func swapContents(listObj []int) {
        for i, j := 0, len(listObj)-1; i < j; i, j = i+1, j-1 {
                listObj[i], listObj[j] = listObj[j], listObj[i]
        }
}
func main() {
    listObj := []int{1, 2, 3}
    swapContents(listObj)
    fmt.Println(listObj)
}
```

The code results in the output:

```
[3 2 1]
```

32. Write a GO Program to find factorial of a given number.

Factorial of a number is the product of multiplication of a number n with every preceding number till it reaches 1. Factorial of 0 is 1.

```
Example:
fact(1) = 1
fact(3) = 3 * 2 * 1 = 6
fact(5) = 5 * 4 * 3 * 2 * 1 = 120
```

Code:

```
package main
import "fmt"
//factorial function
func factorial(n int) int {
    if n == 0 {
        return 1
    }
    return n * factorial(n-1)
}

func main() {
    fmt.Println(factorial(7))
}
```

The output of this code would be:

```
5040
```

33. Write a Go program to find the nth Fibonacci number.

To find the nth Fibonacci number, we have to add the previous 2 Fibonacci numbers as shown below.

```
fib(0)=0
fib(1)=1
fib(2)=1+0 = 1
fib(3)=1+1 = 2
fib(4)=2+1 = 3
:
:
fib(n)=fib(n-1)+fib(n-2)
```

Code:

```
package main
import "fmt"
//nth fibonacci number function
func fibonacci(n int) int {
        if n < 2 {
            return n
        }
        return fibonacci(n-1) + fibonacci(n-2)
}

func main() {
    fmt.Println(fibonacci(7))
}
```

The output of this code would be:

```
13
```

34. Write a Golang code for checking if the given characters are present in a string.

We can do this by using the Contains() method from the strings package.

```
package main

import (
    "fmt"
    "strings"
)

// Main function
func main() {

    // Create and initialize
    string1 := "Welcome to Interviewbit"
    string2 := "Golang Interview Questions"

    //  Check for presence Using Contains() method of strings package
    res1 := strings.Contains(string1, "Interview")
    res2 := strings.Contains(string2, "Go")

    // Displaying the result
    fmt.Println("Is 'Interview' present in string1 : ", res1)
    fmt.Println("Is 'Go' present in string2: ", res2)

}
```

The output of this code is:

```
Is 'Interview' present in string1 :  true
Is 'Go' present in string2:  true
```

35. Write a Go code to compare two slices of a byte.

We can do this by using the Compare() method from the bytes package.

```
package main

import (
    "bytes"
    "fmt"
)

func main() {

    sl1 := []byte{'I', 'N', 'T', 'E', 'R' , 'V', 'I', 'E', 'W'}
    sl2 := []byte{'B', 'I', 'T'}

    // Use Compare function to compare slices
    res := bytes.Compare(sl1, sl2)

    if res == 0 {
        fmt.Println("Equal Slices")
    } else {
        fmt.Println("Unequal Slices")
    }
}
```

The output of this code is:

```
Unequal Slices
```

Conclusion

Golang was developed with the promise of code efficiency for faster software development. Companies have recognized the scope and benefits of Golang and have started to adapt to this language. Some of the notable companies that have already shifted to Golang are Google, Apple, Facebook, Docker, BBC etc. Furthermore, Golang has raised the excitement level of developers in the open-source community as it's been a while since a new language for the backend has been created. Due to these reasons, the scope of Golang is growing rapidly.

According to the data in Golang Cafe from 2021, the average salary of golang developers in India starts from ₹819,565 to ₹1,617,391 per annum. The prospects and benefits are amazing!

References

To Learn Golang:

- Golang Documentation
- Go by Example
- Golang Playground

Links to More Interview Questions

C Interview Questions	Php Interview Questions	C Sharp Interview Questions
Web Api Interview Questions	Hibernate Interview Questions	Node Js Interview Questions
Cpp Interview Questions	Oops Interview Questions	Devops Interview Questions
Machine Learning Interview Questions	Docker Interview Questions	Mysql Interview Questions
Css Interview Questions	Laravel Interview Questions	Asp Net Interview Questions
Django Interview Questions	Dot Net Interview Questions	Kubernetes Interview Questions
Operating System Interview Questions	React Native Interview Questions	Aws Interview Questions
Git Interview Questions	Java 8 Interview Questions	Mongodb Interview Questions
Dbms Interview Questions	Spring Boot Interview Questions	Power Bi Interview Questions
Pl Sql Interview Questions	Tableau Interview Questions	Linux Interview Questions
Ansible Interview Questions	Java Interview Questions	Jenkins Interview Questions