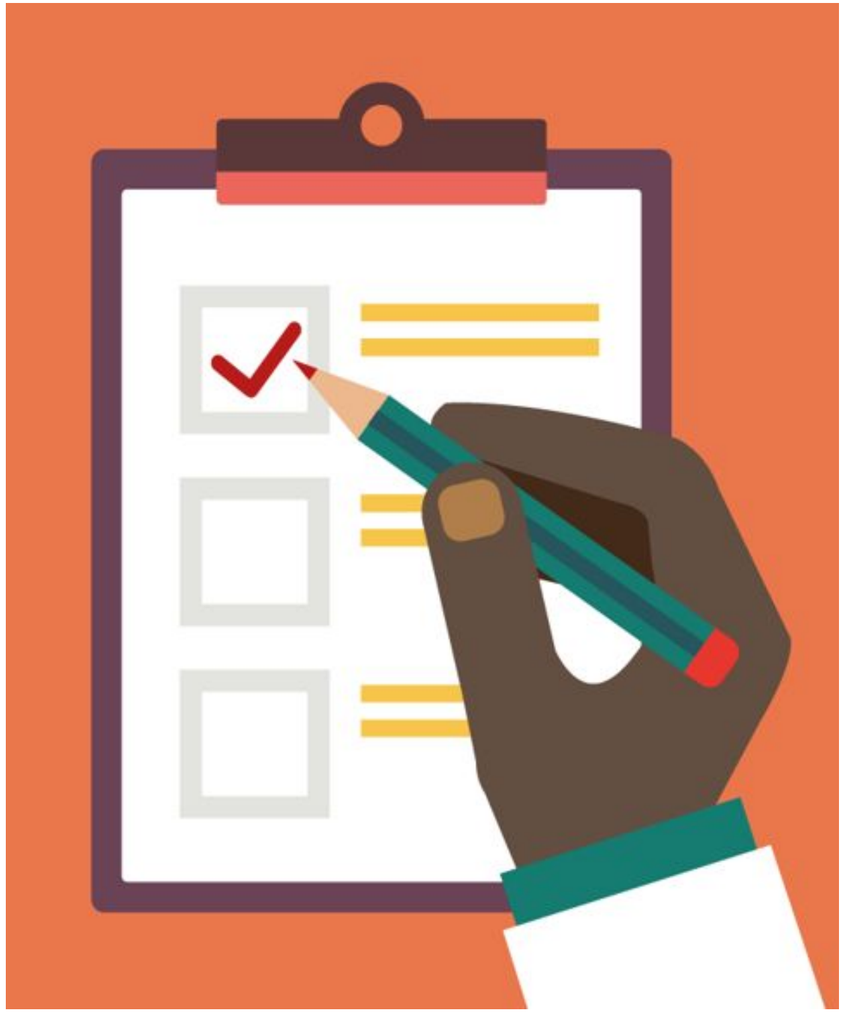
Welcome To

JOSH

DISRUPT YOUR INDUSTRY

Topics to be covered…



- Functions
- Loops
- Conditional Statements
- Switch case

Functions

- What is a function?

  A function is a block of code that performs a specific task. A function takes an input, performs some calculations on the input, and generates an output.

- Advantages of using a function :
  - Reducing duplication of code
  - Decomposing complex problems into simpler pieces
  - Improving clarity of the code
  - Reuse of code
  - Information hiding

Functions

- Functions in Go are created with the **func** keyword
- Function can have any number of input and return parameters.
- **return** keyword is used to return values from functions.
- Functions can be assigned to variables, passed as arguments to functions or returned from functions. This makes the language more flexible.

- Function declaration

  The syntax for declaring a function in go is

```
func functionname(parametername type) returntype {
 //function body
}
```

Functions

```go
package main

import (
    "fmt"
)

func calculateBill(price, no int) int {
    var totalPrice = price * no
    return totalPrice
}

func main() {
    price, no := 90, 6
    totalPrice := calculateBill(price, no)
    fmt.Println("Total price is", totalPrice)
}
```

Run In Go Playground

Functions

Anonymous Function

- An anonymous function is a function that was declared without any named identifier to refer to it.
- Anonymous functions are useful when you want to define a function inline without having to name it.
- You can assign anonymous functions to a variable.
- Go can implement **Closure** using anonymous functions. Ex

```
func main() {

    func() {
        fmt.Println(“Inside anonymous function”)
    }()
}
```

Run In Go Playground

Functions

Variadic Function

- A variadic function can accept variable number of parameters.
- We use the ... (ellipses) operator to define a variadic function.

```
func sum(nums ...int) int {

    res := 0

    for _, n := range nums {
        res += n
    }

    return res
}
```

Run In Go Playground

Loops

Go has only one looping construct, the for loop.

The basic for loop has three components separated by semicolons

- the init statement: executed before the first iteration
- the condition expression: evaluated before every iteration
- the post statement: executed at the end of every iteration

The init statement will often be a short variable declaration, and the variables declared there are visible only in the scope of the for statement.

The loop will stop iterating once the boolean condition evaluates to false

Loops

Simple For loop :

Syntax : **for** initialization; condition; update {

statement(s)

}

```
sum := 0
for i := 1; i < 5; i++ {
    sum += i
}

fmt.Println(sum) // 10 (1+2+3+4)
```

Loops

while loop :

Syntax : **for** condition {
statement(s)
}

```
n := 1
for n < 5 {
    n *= 2
}
fmt.Println(n) // 8 (1*2*2*2)
```

infinite loop :

Syntax : **for** {
statement(s)
}

```
sum := 0
for {
    sum++ // repeated forever
}
fmt.Println(sum) // never reached
```

Loops

For-each range loop :

Looping over elements in slices, arrays, maps, channels or strings is often better done with a range loop.

```
strings := []string{"hello", "world"}

for i, s := range strings {
    fmt.Println(i, s)
}
```

Examples

Exit a loop :

The **break** and **continue** keywords work just as they do in other languages.

```
sum := 0
for i := 1; i < 5; i++ {
    if i%2 != 0 { // skip odd no
        continue
    }
    sum += i
}

fmt.Println(sum) // 6 (2+4)
```

Conditional Statement

Conditional Statement are used to make decision based on the condition

If statement

Go's `if` statements are like its `for` loops

the expression need not be surrounded by parentheses `( )`

but the braces `{ }` are required.

Examples

- If statement
- If else statement
  - If else if statement
- If with a short statement

Conditional Statement

Switch

A switch statement is a shorter way of writing sequence of if else statements.

It runs the first statement whose value is equal to the desired check of condition.

Examples:

- Switch statement
- Switch without an expression

Structs

A struct is a user-defined type that represents a collection of fields.

It can be used in places where it makes sense to group the data into a single unit rather than having each of them as separate values.

Example : structs

Structs Equality

Structs are value types and are comparable if each of their fields are comparable

Two struct variables are considered equal if their corresponding fields are equal

example 1

example 2

THANK YOU