# GO SYNTAX

Line 1:

It contains the package main of the program, which have overall content of the program.It is the initial point to run the program, So it is compulsory to write.

```
1    package main
```

Line 2:

It contains import “fmt”, it is a preprocessor command which tells the compiler to include the files lying in the package.

```
2    import "fmt"
```

Line 3:

main function, it is beginning of execution of program.

```
func main() {

}
```

Line 4:

fmt.Println() is a standard library function to print something as a output on screen.In this, fmt package has transmitted Println method which is used to display the output.

```
func main() {
    fmt.Println("hello world !")
}
```

Single Line Comment:

```
// single line comment
```

Multi line Comment:

```
/* multiline comment */
```

Defining Identifiers:

var variable_name type = expression  // type is optional

- either type or = expression can be omitted, but not both.
- If the = expression is omitted, then the variable value is determined by its type's default value. The default value is usually 0 for numbers ,false for Boolean and "" for string.
- If the type is removed, then the type of the variable is determined by the value-initialize in the expression.

```
var x=5
var y="Menna"
```

• The name of the identifier must begin with a letter or an underscore(_). And the names may contain the letters 'a-z' or 'A-Z' or digits 0-9 as well as the character '_'.
• The name of the identifier should not start with a digit.
• The name of the identifier is case sensitive.
• Keywords is not allowed to use as an identifier name.
• There is no limit on the length of the name of the identifier, but it is advisable to use an optimum length of 4 – 15 letters only.

```
// Valid identifiers:
_geeks23
geeks
gek23sd
Geeks
geeKs
geeks_geeks

// Invalid identifiers:
212geeks
if
default
```

```
var myvariable1 = 20
var myvariable2 = "GeeksforGeeks"
var myvariable3 = 34.80

// Display the value and the
// type of the variables
fmt.Printf("The value of myvariable1 is : %d\n",
                                    myvariable1)

fmt.Printf("The type of myvariable1 is : %T\n",
                                    myvariable1)

fmt.Printf("The value of myvariable2 is : %s\n",
                                      myvariable2)

fmt.Printf("The type of myvariable2 is : %T\n",
                                    myvariable2)

fmt.Printf("The value of myvariable3 is : %f\n",
                                      myvariable3)

fmt.Printf("The type of myvariable3 is : %T\n",
                                    myvariable3)
```

Output:

```
The value of myvariable1 is : 20
The type of myvariable1 is : int
The value of myvariable2 is : GeeksforGeeks
The type of myvariable2 is : string
The value of myvariable3 is : 34.800000
The type of myvariable3 is : float64
```

```
var myvariable1 int
var myvariable2 string
var myvariable3 float64

// Display the zero-value of the variables
fmt.Printf("The value of myvariable1 is : %d\n",
                                    myvariable1)

fmt.Printf("The value of myvariable2 is : %s\n",
                                    myvariable2)

fmt.Printf("The value of myvariable3 is : %f",
                                    myvariable3)
```

Output:

```
The value of myvariable1 is : 0
The value of myvariable2 is :
The value of myvariable3 is : 0.000000
```

- If you use type, then you are allowed to declare multiple variables of the same type in the single declaration

```
var myvariable1, myvariable2, myvariable3 int = 2, 454, 67
```

- If you remove type, then you are allowed to declare multiple variables of a different type in the single declaration. The type of variables is determined by the initialized values.

```
var myvariable1, myvariable2, myvariable3 = 2, "GFG", 67.56
```

Short variable declaration:
The local variables which are declared and initialize in the functions are declared by using short variable declaration.

```
variable_name:= expression
```

:= is a declaration and = is assignment

Using short variable declaration you are allowed to declare multiple variables in the single declaration

```
myvar1, myvar2, myvar3 := 800, 34, 56
```

```
myvar1, myvar2 := 39, 45
myvar3, myvar2 := 45, 100

fmt.Print(myvar1, myvar2, myvar3)
```

Output:

```
39 100 45
```

- Declaring constant value

```
const PI = 3.14
```

- Calculations

```go
func main() {
    p:= 34
    q:= 20

    // Addition
    result1:= p + q
    fmt.Printf("Result of p + q = %d", result1)

    // Subtraction
    result2:= p - q
    fmt.Printf("\nResult of p - q = %d", result2)

    // Multiplication
    result3:= p * q
    fmt.Printf("\nResult of p * q = %d", result3)

    // Division
    result4:= p / q
    fmt.Printf("\nResult of p / q = %d", result4)

    // Modulus
    result5:= p % q
    fmt.Printf("\nResult of p %% q = %d", result5)
}
```

Output:

```
Result of p + q = 54
Result of p - q = 14
Result of p * q = 680
Result of p / q = 1
Result of p % q = 14
```

Relational operators:

```go
func main() {
   p:= 34
   q:= 20

   // '=='(Equal To)
   result1:= p == q
   fmt.Println(result1)

   // '!='(Not Equal To)
   result2:= p != q
   fmt.Println(result2)

   // '<'(Less Than)
   result3:= p < q
   fmt.Println(result3)

   // '>'(Greater Than)
   result4:= p > q
   fmt.Println(result4)

   // '>='(Greater Than Equal To)
   result5:= p >= q
   fmt.Println(result5)

   // '<='(Less Than Equal To)
   result6:= p <= q
   fmt.Println(result6)
```

Output:

```
false
true
false
true
true
false
```

Logical operator:

```
func main() {
    var p int = 23
    var q int = 60

    if(p!=q && p<=q){
        fmt.Println("True")
    }

    if(p!=q || p<=q){
        fmt.Println("True")
    }

    if(!(p==q)){
        fmt.Println("True")
    }

}
```

Output:

```
True
True
True
```

Assignment operator:

```
func main() {
   var p int = 45
    var q int = 50

   // "="(Simple Assignment)
   p = q
   fmt.Println(p)

   // "+="(Add Assignment)
    p += q
   fmt.Println(p)

   //"-="(Subtract Assignment)
   p-=q
   fmt.Println(p)

   // "*="(Multiply Assignment)
   p*= q
   fmt.Println(p)

   // "/="(Division Assignment)
    p /= q
   fmt.Println(p)

    // "%="(Modulus Assignment)
    p %= q
   fmt.Println(p)

}
```

Output:

```
50
100
50
2500
50
0
```

Ex:

```
const A = "GFG"
var B = "GeeksforGeeks"

// Concat strings.
var helloWorld = A+ " " + B
helloWorld += "!"
fmt.Println(helloWorld)

// Compare strings.
fmt.Println(A == "GFG")
fmt.Println(B < A)
```

Output:

```
GFG GeeksforGeeks!
true
false
```

If statement:

```
if condition {

    // Statements to execute if
    // condition is true
}
```

```
func main() {

   // taking a local variable
   var v int = 700

   // using if statement for
   // checking the condition
   if v < 1000 {

      // print the following if
      // condition evaluates to true
      fmt.Printf("v is less than 1000\n")
   }

   fmt.Printf("Value of v is : %d\n", v)
```

Output:

```
v is less than 1000
value of v is : 700
```

If …. Else statement:

```
if condition {

    // Executes this block if
    // condition is true
} else {

    // Executes this block if
    // condition is false
}
```

```
func main() {

   // taking a local variable
   var v int = 1200

   // using if statement for
   // checking the condition
   if v < 1000 {

      // print the following if
      // condition evaluates to true
      fmt.Printf("v is less than 1000\n")

   } else {

       // print the following if
      // condition evaluates to true
      fmt.Printf("v is greater than 1000\n")
   }

}
```

Output:

```
v is greater than 1000
```

If else if

```
if condition_1 {

     // this block will execute
     // when condition_1 is true

} else if condition_2 {

    // this block will execute
    // when condition2 is true
}
.
.
. else {

      // this block will execute when none
     // of the condition is true
}
```

```
func main() {

   // taking a local variable
   var v1 int = 700

   // checking the condition
   if v1 == 100 {

      // if condition is true then
      // display the following */
      fmt.Printf("Value of v1 is 100\n")

   } else if v1 == 200 {

      fmt.Printf("Value of a is 20\n")

   } else if v1 == 300 {

      fmt.Printf("Value of a is 300\n")

   } else {

      // if none of the conditions is true
      fmt.Printf("None of the values is matching\n")
   }
}
```

Output:

```
None of the values is matching
```

Looping

Simple for loop

```
for initialization; condition; post{
        // statements....
}
```

- The initialization statement is optional and executes before for loop starts. The initialization statement is always in a simple statement like variable declarations, increment or assignment statements, or function calls.
- The condition statement holds a boolean expression, which is evaluated at the starting of each iteration of the loop. If the value of the conditional statement is true, then the loop executes.
- The post statement is executed after the body of the for-loop. After the post statement, the condition statement evaluates again if the value of the conditional statement is false, then the loop ends.

Ex

```
// for loop
// This loop starts when i = 0
// executes till i<4 condition is true
// post statement is i++
for i := 0; i < 4; i++{
  fmt.Printf("GeeksforGeeks\n")
}
```

Output

```
GeeksforGeeks
GeeksforGeeks
GeeksforGeeks
GeeksforGeeks
```

Infinite loop

```
for{
     // Statement...
}
```

Ex

```
// Infinite loop
for {
  fmt.Printf("GeeksforGeeks\n")
}
```

Output

```
GeeksforGeeks
GeeksforGeeks
GeeksforGeeks
GeeksforGeeks
GeeksforGeeks
GeeksforGeeks
GeeksforGeeks
GeeksforGeeks
GeeksforGeeks
GeeksforGeeks
...........
```

For loop as while loop

```
for condition{
    // statement..
}
```

Ex

```
  // while loop
  // for loop executes till
  // i < 3 condition is true
  i:= 0
  for i < 3 {
     i += 2
  }
fmt.Println(i)
```

Output

```
 4
```

Simple range in for loop

```
for i, j:= range rvariable{
   // statement..
}
```

- i and j are the variables in which the values of the iteration are assigned. They are also known as iteration variables.
- The second variable, i.e, j is optional.
- The range expression is evaluated once before the starting of the loop.

Ex

```
// Here rvariable is a array
rvariable:= []string{"GFG", "Geeks", "GeeksforGeeks"}

// i and j stores the value of rvariable
// i store index number of individual string and
// j store individual string of the given array
for i, j:= range rvariable {
   fmt.Println(i, j)
}
```

Output

```
0 GFG
1 Geeks
2 GeeksforGeeks
```

Using for loop for string

A for loop can iterate over the Unicode code point for a string

```
for index, chr:= range str{
     // Statement..
}
```

Ex

```
// String as a range in the for loop
for i, j:= range "XabCd" {
   fmt.Printf("The index number of %U is %d\n", j, i)
}
```

Output

```
The index number of U+0058 is 0
The index number of U+0061 is 1
The index number of U+0062 is 2
The index number of U+0043 is 3
The index number of U+0064 is 4
```

## Go language supports two types of switch statements:

1. **Expression Switch**
2. **Type Switch**

### Expression Switch

```
switch optstatement; optexpression{
case expression1: Statement..
case expression2: Statement..
...
default: Statement..
}
```

- Both optstatement and optexpression in the expression switch are optional statements.
- If both optstatement and optexpression are present, then a semi-colon(;) is required in between them.
- If the switch does not contain any expression, then the compiler assume that the expression is true.
- The optional statement, i.e, optstatement contains simple statements like variable declarations, increment or assignment statements, or function calls, etc.
- If a variable present in the optional statement, then the scope of the variable is limited to that switch statement.

- In switch statement, the case and default statement does not contain any break statement. But you are allowed to use break and fall through statement if your program required.
- The default statement is optional in switch statement.
- If a case can contain multiple values and these values are separated by comma(,).
- If a case does not contain any expression, then the compiler assume that the expression is true.

```
// Switch statement with both
// optional statement, i.e, day:=4
// and expression, i.e, day
switch day:=4; day{
   case 1:
   fmt.Println("Monday")
   case 2:
   fmt.Println("Tuesday")
   case 3:
   fmt.Println("Wednesday")
   case 4:
   fmt.Println("Thursday")
   case 5:
   fmt.Println("Friday")
   case 6:
   fmt.Println("Saturday")
   case 7:
   fmt.Println("Sunday")
   default:
   fmt.Println("Invalid")
}
```

Output

```
Thursday
```

Ex2

```
 var value int = 2

 // Switch statement without an
 // optional statement and
 // expression
switch {
    case value == 1:
    fmt.Println("Hello")
    case value == 2:
    fmt.Println("Bonjour")
    case value == 3:
    fmt.Println("Namstay")
    default:
    fmt.Println("Invalid")
}
```

Output

```
Bonjour
```

Ex3

```
 var value string = "five"

 // Switch statement without default statement
 // Multiple values in case statement
switch value {
    case "one":
    fmt.Println("C#")
    case "two", "three":
    fmt.Println("Go")
    case "four", "five", "six":
    fmt.Println("Java")
}
```

Output

```
Java
```

## Type Switch

Type switch is used when you want to compare types. In this switch, the case contains the type which is going to compare with the type present in the switch expression

```
switch optstatement; typeswitchexpression{
case typelist 1: Statement..
case typelist 2: Statement..
...
default: Statement..
}
```

- The optional statement, i.e., optstatement is similar as in the switch expression.
- If a case can contain multiple values and these values are separated by comma(,).
- In type switch statement, the case and default statement do not contain any break statement. But you are allowed to use break and fall through statement if your program required.
- The default statement is optional in type switch statement.
- The type switch expression is an expression whose result is a type.
- If an expression is assigned in type switch expression using := operator, then the type of that variable depends upon the type present in case clause. If the case clause contains two or more types, then the type of the variable is the type in which it is created in type switch expression.

```
var value interface{}
switch q:= value.(type) {
    case bool:
    fmt.Println("value is of boolean type")
    case float64:
    fmt.Println("value is of float64 type")
    case int:
    fmt.Println("value is of int type")
    default:
    fmt.Printf("value is of type: %T", q)

}
```

Output

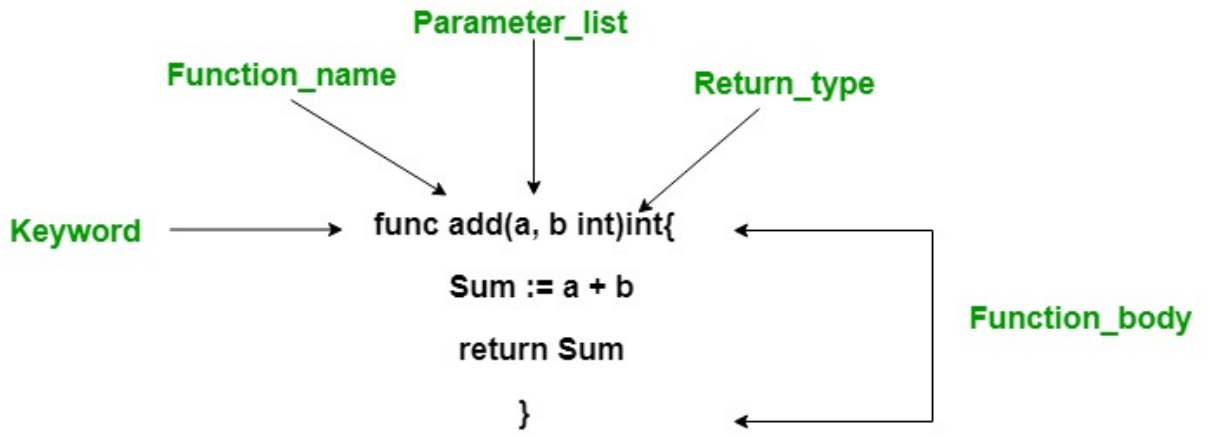
```
value is of type: <nil>
```

Functions

```
func function_name(Parameter-list)(Return_type){
    // function body.....
}
```

- func:
  It is a keyword in Go language, which is used to create a function.
- function_name:
  It is the name of the function.
- Parameter-list:
  It contains the name and the type of the function parameters.
- Return_type:
  It is optional and it contain the types of the values that function returns. If you are using return_type in your function, then it is necessary to use a return statement in your function.

Ex




```go
// area() is used to find the
// area of the rectangle
// area() function two parameters,
// i.e, length and width
func area(length, width int)int{

    Ar := length* width
    return Ar
}

// Main function
func main() {

   // Display the area of the rectangle
   // with method calling
   fmt.Printf("Area of rectangle is : %d", area(12, 10))
}
```

Output

```
Area of rectangle is : 120
```

Anonymous Function

An anonymous function is a function which doesn't contain any name. It is useful when you want to create an inline function. In Go language, an anonymous function can form a closure. An anonymous function is also known as function literal.

```
func(parameter_list)(return_type){
// code..

// Use return statement if return_type are given
// if return_type is not given, then do not
// use return statement
return
}()
```

```
func main() {

    // Anonymous function
   func(){

      fmt.Println("Welcome! to GeeksforGeeks")
  }()

}
```

Output

```
 Welcome! to GeeksforGeeks
```

In Go language, you are allowed to assign an anonymous function to a variable. When you assign a function to a variable, then the type of the variable is of function type and you can call that variable

```
func main() {

    // Assigning an anonymous
   // function to a variable
   value := func(){
      fmt.Println("Welcome! to GeeksforGeeks")
  }
  value()

}
```

Output

```
Welcome! to GeeksforGeeks
```

You can also pass arguments in the anonymous function

```
  // Passing arguments in anonymous function
func(ele string){
    fmt.Println(ele)
}("GeeksforGeeks")
```

Output

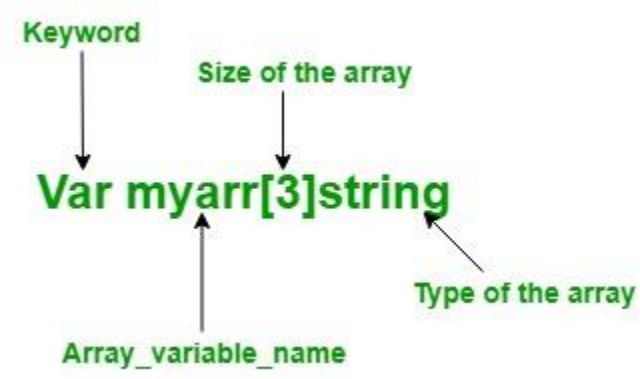
```
 GeeksforGeeks
```

Arrays

```
Var array_name[length]Type
```

arrays are mutable, so that you can use array[index] syntax to the left-hand side of the assignment to set the elements of the array at the given index

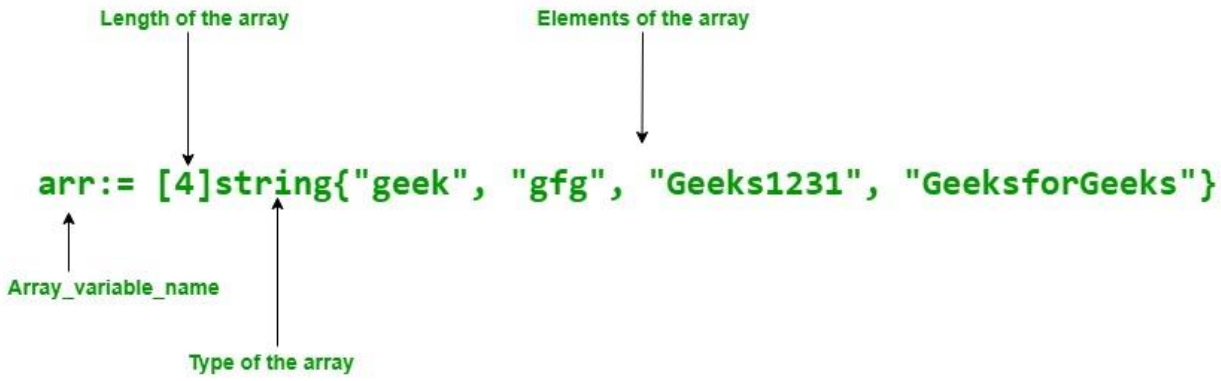
```
Var array_name[index] = element
```




- You can access the elements of the array by using the index value or by using for loop.
- In Go language, the array type is one-dimensional.
- The length of the array is fixed and unchangeable.
- You are allowed to store duplicate elements in an array.

Using shorthand declaration

```
array_name:= [length]Type{item1, item2, item3,...itemN}
```




```
// Shorthand declaration of array
arr:= [4]string{"geek", "gfg", "Geeks1231", "GeeksforGeeks"}

// Accessing the elements of
// the array Using for loop
fmt.Println("Elements of the array:")

for i:= 0; i < 3; i++{
fmt.Println(arr[i])
}
```

```
Elements of the array:
geek
gfg
Geeks1231
```

Multi dimensional array

```
Array_name[Length1][Length2]..[LengthN]Type
```

```go
// Creating and initializing
// 2-dimensional array
// Using shorthand declaration
// Here the (,) Comma is necessary
arr := [3][3]string{{"C #", "C", "Python"}, {"Java", "Scala", "Perl"},
    {"C++", "Go", "HTML"}}

// Accessing the values of the
// array Using for loop
fmt.Println("Elements of Array 1")
for x := 0; x < 3; x++ {
    for y := 0; y < 3; y++ {

        fmt.Println(arr[x][y])
    }
}

// Creating a 2-dimensional
// array using var keyword
// and initializing a multi
// -dimensional array using index
var arr1 [2][2]int
arr1[0][0] = 100
arr1[0][1] = 200
arr1[1][0] = 300
arr1[1][1] = 400

// Accessing the values of the array
fmt.Println("Elements of array 2")
for p := 0; p < 2; p++ {
    for q := 0; q < 2; q++ {
        fmt.Println(arr1[p][q])
    }
}
```

```
Elements of Array 1
C#
C
Python
Java
Scala
Perl
C++
Go
HTML
Elements of array 2
100
200
300
400
```

In an array, if an array does not initialize explicitly, then the default value of this array is 0.

```
// Creating an array of int type
// which stores, two elements
// Here, we do not initialize the
// array so the value of the array
// is zero
var myarr[2] int fmt.Println("Elements of the Array: ", myarr)
```

```
Elements of the Array : [0 0]
```

In an array, you are allowed to iterate over the range of the elements of the array

```
// Creating an array whose size
// is represented by the ellipsis
myarray:= [...]int{29, 79, 49, 39,
                   20, 49, 48, 49}

// Iterate array using for loop
for x:=0; x < len(myarray); x++{
fmt.Printf("%d\n", myarray[x])
}
```

```
29
79
49
39
20
49
48
49
```