**Go consist of following things.**

1. Package declaration
2. Import packages
3. Functions
4. variables
5. Statements and expression
6. Comments

Example explained

**Line 1:** In Go, every program is part of a package. We define this using the package keyword. In this example, the program belongs to the main package.

**Line 2:** import ("fmt") lets us import files included in the fmt package.

**Line 3:** A blank line. Go ignores white space. Having white spaces in code makes it more readable.

**Line 4:** func main() {} is a function. Any code inside its curly brackets {} will be executed.

**Line 5:** fmt.Println() is a function made available from the fmt package. It is used to output/print text. In our example it will output "Hello World!".

fmt.Println("Hello World!") is a statement.

In Go, statements are separated by ending a line (hitting the Enter key) or by a semicolon ";".

Hitting the Enter key adds ";" to the end of the line implicitly (does not show up in the source code).

The left curly bracket { cannot come at the start of a line.

**Declaring and creating a variable in Golang.**

**1 -🡪 *var*variablename type*=*value**

**2 🡪 variablename := value**

## **Difference Between var and :=**

There are some small differences between the var var :=:

|  |  |
| --- | --- |
| **var** | **:=** |
| Can be used **inside**and **outside** of functions | Can only be used **inside** functions |
| Variable declaration and value assignment **can be done separately** | Variable declaration and value assignment **cannot be done separately** (must be done in the same line) |

Declaring multiple variable and value in same line.

**Note:** If you use the type keyword, it is only possible to declare **one type** of variable per line.

**myVariableName = "John" called as camel case. -- > first word is capital, except the first word.**

MyVariableName = "John" called as pascal case. 🡪 every word start with capital letter.

my\_variable\_name = "John" called as snake case. -🡪 words are separated through \_ .

If a variable should have a fixed value that cannot be changed, you can use the const keyword.

The const keyword declares the variable as "constant", which means that it is **unchangeable and read-only**.

***const* CONSTNAME type *=* value**

**constant rules :**

 Constant names are usually written in uppercase letters (for easy identification and differentiation from variables)

 Constants can be declared both inside and outside of a function.

**Formatting in GO.**

%v -🡪 value  
%T 🡪 Type  
rest other refer w3 school.

Go data type:  
Bool : True or false -🡪typed and untype  
int: singed & unsigned -🡪 +ve & -ve, and only positive value.  
strings: 🡪 “value declare in quotes. “   
float: 🡪 values in decimals.

Go data structure.

**Array :  
1**

**Var array\_name = [length] datatype { …..}  
var array\_name = [….] datatype {…..} // length has inferred from the compiler.**

**2**

**array\_name := [length] datatype {……}  
array\_name := [….] datatype {…..}**

**Slice … new data structure.**

1. **Similar to the array.**
2. **Adv it can expand and shrink, more powerful than array.**
3. **Cap is used to find the max capacity to which the slice can grow.**

**Declaration of slice.**

**Myslice := [ ]int { ..}**

**Creating a slice from the array.**

**Eg:**

**Var array\_1 = [length]data\_type{value}  
myslice = array\_1[start:end]**

**Create a slice from make function.**

**slice\_name := make([]type, length, capacity)**

**modify the slice.**

**Accessing the slice through the indexes is similar to that of array.  
changing the value of slice through the index is similar to that of array.  
Append the slice:**

**Slice\_name = []int{slice\_name, element1, element2}**

**Append slice  
Change the size through the slice.**

**Copy function. To save the memory function.**copy(dest, src)

Scope of variable in GO.

Global variable: var declared globally (outside block or function) and can be used throughout the program.

Local variable: defined in the block or function, can be used with in the block by any loop, and if **same name is declared, within the scope twice, it will be compile time error**.

**Note: What will happen there exists a local variable with the same name as that of the global variable inside a function?**

**The answer is simple i.e. compiler will give preference to the local variable. Usually when two variable with the same name is defined then the compiler produces a compile-time error. But if the variables are defined in different scopes then the compiler allows it. Whenever there is a local variable defined with the same name as that of a global variable then the compiler will give precedence to the local variable.**

Var in go

**If you removed the expression then the variable will contain the zero-value for numbers and false for booleans “” for strings and nil for interface and reference type by default. So, there is no such concept of an uninitialized variable in Go language.**

**Loop in go**

If the array, string, slice, or map is empty, then for loop does not give an error and continue its flow. Or in other words, if the array, string, slice, or map is nil then the number of iterations of the for loop is zero.