# Go Language

**What is Go?**

**-** Go is an open source programming language that makes it simple to build secure, scalable systems

- Go, also called as Golang or Go language.

- Go is known for its efficiency, strong concurrency mechanisms, and straightforward syntax.

- Used for Backend Development.

## **What is Go Used For?**

- Web Development (server-side)

- Developing network-based programs

- Developing cross-platform enterprise application

- cloud-native development

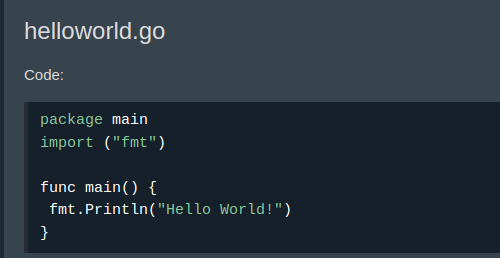
**Go file consists of the following parts**:

- Package declaration

- Import packages

- Functions

- Statements and expressions

**Sample Program**:

* In Go, any executable code belongs to the main package.
* 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.

**Comments**

* **Singleline comments**

// this is single line commet

* **Multiline comments**

*/\* This is multiline comment \**/

**Vriable Types**

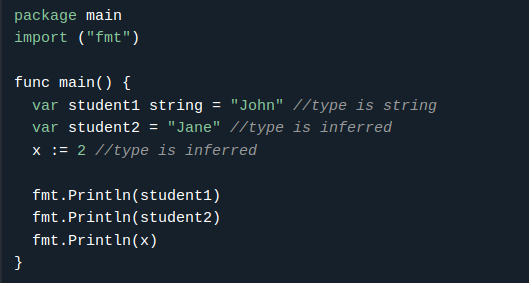
* int
* float32
* string
* bool

**Vriable Declaration**

1. using var keyword

Ex: var variablename type = value (we can pass either type or value or both)ue

1. using := sign

Ex: variablename := value (here type of variable will de inferred from value)

**Multiple variable declaration:**

Ex: var a, b, c, d int = 1, 2, 3, 4

var a, name, flag = 1, “XYZ”, true

OR

var (

a int = 1

b string = “xyz”

)

**Naming Convention rules:**

* should be alphanumeric (a-z, A-Z, 0-9 and \_ )
* should start with letter or an undesrcore (\_)
* it is case sensitive
* should not contain spaces and should not be any GO keywords

**Output Functions**

1. Print()

- this will print as it is same format no space, no new line will add

1. Printf()

- this function first formats its argument based on the given formatting verb and then prints them

Ex:

var i = 1

Printf("i has value: %v and type: %T\n", i, i)

// %v gives value, %T gives type of variable

1. Println() ( this will add a whitespace between the arguments, and a newline is added at the end )

**Formatting Verbs**

**-** we can use below mentioned formatting verbs with printf() function

1. %v ( Prints value )
2. %T (Prints the type of the value)
3. %#v ( Prints the value with Go-syntax format )
4. %% (Prints the % sign)
5. https://www.w3schools.com/go/go\_formatting\_verbs.php

**Datatypes**

Go has three basic data types

* bool: it is either true or false
* Numeric: represents integer types, floating point values and complex types
* String: represents string values

**Arrays**

1. declare array using var

Ex: var array\_name = [length]datatype{values}

OR

var array\_name = [...]datatype{values} (in this length of the array will be infrerred from the values)

1. declare array using :=

Ex: array\_name := [length]datatype{values}

OR

array\_name := [...]datatype{values}

**Slices**

-slices are similar to arrays but the length of a slice can grow and shrink as you see fit.

- three different ways to create slices

1. []datatype{values}

syntax: myslice1 := []int{} OR myslice1 := []int{values}

1. create from an array

syntax: array := [...]int{values}

myslice = array[start:end]

1. using the make() function

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

- Append element to the slice

syntax: slice\_name = append(slice\_name, element1, element2, ...)

**Oprators**

1. Arithmetic Operators

+, -, \*, /, %, ++, --

1. Assigment Operators

=, +=, -=, \*=, /=, %=, &=, |=, ^=, >>=, <<=

1. comparision Operators

==, >=, <=, !=, >, <

1. Logical Operators

&&, ||, !

1. Bitwise Operators

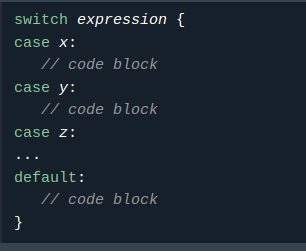
&, |, ^, <<, >>

**Conditional Statements**

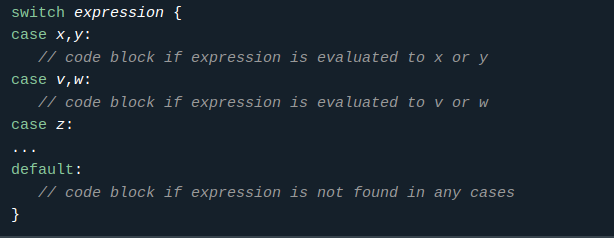
-if, else, else if, switch

**Switch Statements**

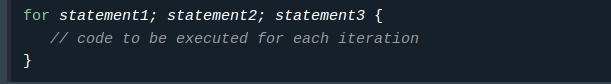
1. Single Switch case



1. Multi Switch case



**For loop**

****

**statement1** Initializes the loop counter value.

**Statement2** Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.

**Statement3** Increases the loop counter value.

**Continue**

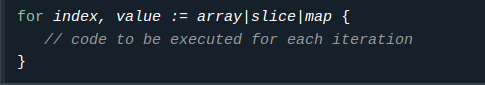
- if want to skip one or some steps then we can use continue keyword

**Break**

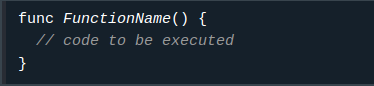
- used to terminate loop execution

**Range**

- Range keyword is used to more easily iterate over an array, slice or map. It returns both the index and the value

****

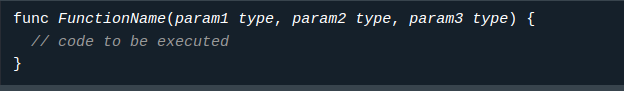
**Functions**

Syntax:

Naming Convention:

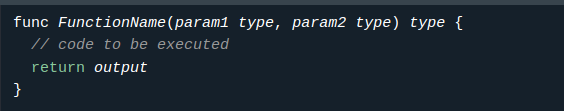
1. must start with a letter
2. can only contain alpha-numeric characters (a-z,A-Z,\_)
3. case sensitive
4. can not contain spaces

**Functions Parameters and arguments**

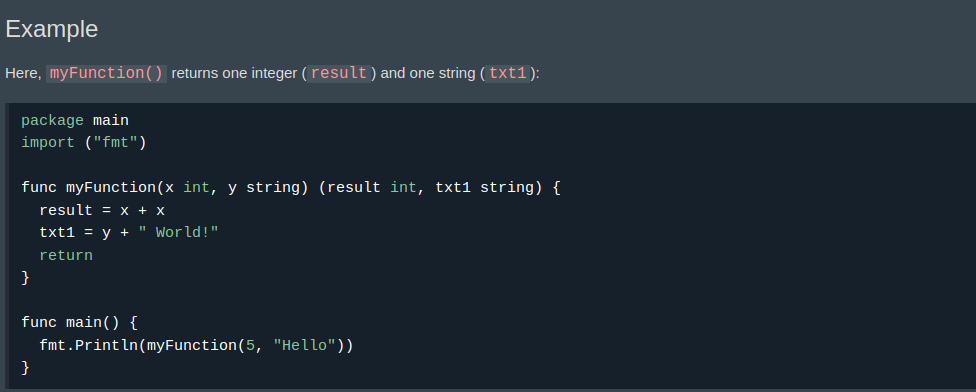
****

**Function Returns**

- If you want the function to return a value, you need to define the data type of the return value with return keyword



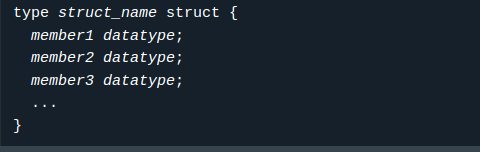
- Go functions can also return multiple values.



**Struct / Structure**

- Struct is used to create a collection of members of different data types, into a single variable.

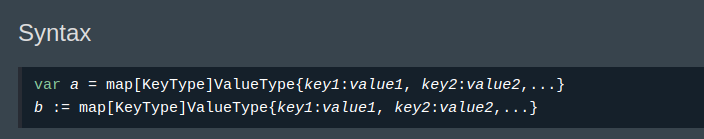
- in a array we can store same type of elements



**Maps**

* Maps are used to store data values in key:value pairs.
* It is an unordered and changeable collection that does not allow duplicates.
* Below are mentioned ways to create map

1. using var and :=



2. using make() function

