## Go Syntax

A Go file consists of the following parts:

- Package declaration
- Import packages
- Functions
- Statements and expressions

Look at the following code, to understand it better:

Example

```
package main
import ("fmt")

func main() {
  fmt.Println("Hello World!")
}
```

## Hello World Example

A Go program basically consists of the following parts −

- Package Declaration
- Import Packages
- Functions
- Variables
- Statements and Expressions
- Comments

Let us look at a simple code that would print the words "Hello World" −

```
package main
import "fmt"
func main() {
   /* This is my first sample program. */
   fmt.Println("Hello, World!")
}
```

Let us take a look at the various parts of the above program −

- The first line of the program package main defines the package name in which this program should lie. It is a mandatory statement, as Go programs run in packages. The main package is the starting point to run the program. Each package has a path and name associated with it.
- The next line import "fmt" is a preprocessor command which tells the Go compiler to include the files lying in the package fmt.
- The next line func main() is the main function where the program execution begins.
- The next line /*...*/ is ignored by the compiler and it is there to add comments in the program. Comments are also represented using // similar to Java or C++ comments.
- The next line fmt.Println(...) is another function available in Go which causes the message "Hello, World!" to be displayed on the screen. Here the fmt package has exported the Println method which is used to display the message on the screen.

- Notice the capital P of the Println method. In Go language, a name is exported if it starts with a capital letter. Exported means the function or variable/constant is accessible to the importer of the respective package.

## Executing a Go Program

Let us discuss how to save the source code in a file, compile it, and finally execute the program. Please follow the steps given below −

- Open a text editor and add the above-mentioned code.
- Save the file as *hello.go*
- Open the command prompt.
- Go to the directory where you saved the file.
- Type go run *hello.go* and press enter to run your code.
- If there are no errors in your code, then you will see *"Hello World!"* printed on the screen.

```
$ go run hello.go
Hello, World!
```

## Go Comments

Single-line comments start with two forward slashes (//).
Multi-line comments start with /* and ends with */.
Example

```
package main
import ("fmt")

func main() {
  /* The code below will print Hello World
  to the screen, and it is amazing */
  fmt.Println("Hello World!")
}
```

## Go Variable Types

In Go, there are different types of variables, for example:

- int- stores integers (whole numbers), such as 123 or -123
- float32- stores floating point numbers, with decimals, such as 19.99 or -19.99
- string - stores text, such as "Hello World". String values are surrounded by double quotes
- bool- stores values with two states: true or false

Declaring (Creating) Variables
In Go, there are two ways to declare a variable:
1. With the var keyword:
Use the var keyword, followed by variable name and type:
Syntax

```
var variablename type = value
```

2. With the := sign:
Use the := sign, followed by the variable value:
Syntax

```
variablename := value
```

## Go Variable Declaration in a Block

Multiple variable declarations can also be grouped together into a block for greater readability:
Example

```
package main
import ("fmt")

func main() {
  var (
        a int
        b int = 1
        c string = "hello"
  )

 fmt.Println(a)
 fmt.Println(b)
 fmt.Println(c)
}
```

## Go Constants

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.
Syntax

```
const CONSTNAME type = value
```

Declaring a Constant
Here is an example of declaring a constant in Go:
Example

```
package main
import ("fmt")

const PI = 3.14

func main() {
 fmt.Println(PI)
}
```

## Go Data Types

Data type is an important concept in programming. Data type specifies the size and type of variable values.

Go is statically typed, meaning that once a variable type is defined, it can only store data of that type.

Go has three basic data types:

- bool: represents a boolean value and is either true or false
- Numeric: represents integer types, floating point values, and complex types
- string: represents a string value

Example

This example shows some of the different data types in Go:

```
package main
import ("fmt")

func main() {
  var a bool = true     // Boolean
  var b int = 5         // Integer
  var c float32 = 3.14  // Floating point number
  var d string = "Hi!"  // String

  fmt.Println("Boolean: ", a)
  fmt.Println("Integer: ", b)
  fmt.Println("Float:   ", c)
  fmt.Println("String:  ", d)
}
```

## Declare an Array

In Go, there are two ways to declare an array:

1. With the var keyword:

Syntax

```
var array_name = [length]datatype{values} // here length is defined

or

var array_name = [...]datatype{values} // here length is inferred
```

2. With the := sign:

Syntax

```
array_name := [length]datatype{values} // here length is defined

or

array_name := [...]datatype{values} // here length is inferred
```

Example

This example declares two arrays (arr1 and arr2) with defined lengths:

```
package main
import ("fmt")
```

```
func main() {
  var arr1 = [3]int{1,2,3}
  arr2 := [5]int{4,5,6,7,8}

  fmt.Println(arr1)
  fmt.Println(arr2)
}
```
Result:
```
[1 2 3]
[4 5 6 7 8]
```

## Go Slices

Slices are similar to arrays, but are more powerful and flexible.
Like arrays, slices are also used to store multiple values of the same type in a single variable.
However, unlike arrays, the length of a slice can grow and shrink as you see fit.
In Go, there are several ways to create a slice:

- Using the []*datatype*{*values*} format
- Create a slice from an array
- Using the make() function

Create a Slice With []datatype{values}
Syntax
```
slice_name := []datatype{values}
```

A common way of declaring a slice is like this:
```
myslice := []int
```

To initialize the slice during declaration, use this:
```
myslice := []int{1,2,3}
```

Example
This example shows how to create slices using the []datatype{values} format:
```
package main

import ("fmt")
func main() {
  myslice1 := []int{}
  fmt.Println(len(myslice1))
  fmt.Println(cap(myslice1))
  fmt.Println(myslice1)
  myslice2 := []string{"Go", "Slices", "Are", "Powerful"}
  fmt.Println(len(myslice2))
  fmt.Println(cap(myslice2))
  fmt.Println(myslice2)
}
```

Result:
```
0
0
[]
4
4
```

## The else if Statement

Use the else if statement to specify a new condition if the first condition is false.

Syntax
```
if condition1 {
  // code to be executed if condition1 is true
} else if condition2 {
  // code to be executed if condition1 is false and condition2 is true
} else {
  // code to be executed if condition1 and condition2 are both false
}
```

## The switch Statement

Use the switch statement to select one of many code blocks to be executed.
The switch statement in Go is similar to the ones in C, C++, Java, JavaScript, and PHP. The difference is that it only runs the matched case so it does not need a break statement.
Single-Case switch Syntax

Syntax
```
switch expression {
case x:
  // code block
case y:
  // code block
case z:
...
default:
  // code block
}
```
for Loop Examples

Example 1
This example will print the numbers from 0 to 4:

```
package main

import ("fmt")

func main() {
  for i:=0; i < 5; i++ {
    fmt.Println(i)
  }
}
```

Result:

```
0
1
2
3
4
```

## Return Values

If you want the function to return a value, you need to define the data type of the return value (such as int, string, etc), and also use the return keyword inside the function:

Syntax

```
func FunctionName(param1 type, param2 type) type {
  // code to be executed
  return output
}
```

## Function Return Example

Example
Here, myFunction() receives two integers (x and y) and returns their addition (x + y) as integer (int):

```
package main
import ("fmt")

func myFunction(x int, y int) int {
  return x + y
}

func main() {
  fmt.Println(myFunction(1, 2))
}
```

Result:

```
3
```