Golanng

1. Go Exercise - hello world

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

Explanation line by line

1. line: in GO each program is part of a package, we define this using **package** as a keyword in this example the program belongs to the **main** or principal package = ***package main***
2. line: **import ("fmt")** lets us import files included in the **fmt** package
3. line: a blank space in GO is ignored but it looks more readable for a user
4. line: func main() {} is a function, any code inside the braces will be executed
5. line: fmt.Println() is a function made possible from the **fmt** package, it is used in the example, this will print "hello world"
6. EXTRA: in GO any executable belongs to the main package

2. Declarations in GO

- fmt.Println("Hello World") is a declaration
- in go the declarations are separated by the end of line pressing the enter key or putting a semicolon ; pressing enter adds a semicolon at the end of the sentence (not shown in the final code) braces cannot be placed at the beginning of a line

3. Compact code in GO

- you can write more compact code in go as shown in the example

```
package main; import ("fmt"); func main () { fmt.Println("hello world");}
```

4. Comments in GO

- a comment is text that is ignored during execution
- comments make the code more readable and are usually used to explain it
- they are also useful to prevent the execution of code when testing an alternative code
- GO allows single-line or multi-line comments

5. Single-line comments

- comments start with double slashes (//)
- any code that is between (//) will not be executed

```
//this is a comment
package main
import ("fmt")
func main (){
    //this is a comment
    fmt.Print("hello world")
}
```

6. Multi-line comments

- each comment in a multiple line starts with /* and ends with */
- all text between these characters will be ignored

```
package main
import ("fmt")

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

7. Variables

- in GO we have several types of variants for example:

  > int: stores integer numbers such as 123 or -321
  > float32: stores numbers with decimals for example 19.99 or -19.99
  > string: stores text like "hello world", string values are between quotes
  > bool: stores booleans: true and false

8. Declarations or creation of variables

- In go there are two ways to declare a variable
    1. using the keyword "var"

```
var variable_name type = value
```

2. with the character :=

```
variable_name := value
```

9. Declaration of variables with an initial value

- if the value of a variable is known from the beginning you can declare and assign it in one line

```
package main
import ("fmt")

func main(){
  var student1 string = "john"
  var student2 string = "jane"
  x := 2
  fmt.Println(student1)
  fmt.Println(student2)
  fmt.Println(x)
}
```

10. Value assigned after declaration

- it is possible to assign a value to a variable after declaring it, this is very useful in cases where the variable is not known

```
package main
import("fmt")

func main() {
  var student1 string
  student1 = "John"
  fmt.Println(student1)
}
```

11. Differences between var and :=

- var: can be used inside or outside functions, the declaration of the variable and the assigned value can be done separately
- := : can only be used inside functions, the variable declaration cannot be done separately

```
package main
import ("fmt")

var a int
var b int = 2
var c = 3
```

```
func main() {
  a = 1
  fmt.Println(a)
  fmt.Println(b)
  fmt.Println(c)
}
```

12. Multiple declarations in GO

- in GO it is possible to declare multiple variables in the same line

```
package main
import ("fmt")

func main(){
  var a, b, c, d int = 1, 2, 3, 4
  fmt.Println(a)
  fmt.Println(b)
  fmt.Println(c)
  fmt.Println(d)
}
```

- if the keyword type is not specified you can declare different types of variables at the same time

```
package main
import("fmt")

func main() {
  var a, b = 6, "hello"
  c, d := 7, "world"
  fmt.Println(a)
  fmt.Println(b)
  fmt.Println(c)
  fmt.Println(d)
}
```

13. Declaration of variables in block

```
package main
import ("fmt")

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

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

14. The rules for variable names in GO

- a variable can have a short name such as (x or y) or a more descriptive one such as (age, price, name, etc.)

> the rules for variables are:
>
> > the variable must start with a letter or underscore (_)
>
> > a variable cannot start with a number
>
> > a variable can only contain alphanumeric characters and underscores (a-z A-Z 0-9 and _)
>
> > the name of the variables is case-sensitive, it's not the same (age, Age, AGE)
>
> > there is no length limit in the variable name
>
> > a variable cannot contain spaces
>
> > the variable name cannot be any of GO's keywords

Multi-word variable names

- these are some of the techniques you can use to make variables more readable
- Pascal style

```
MyVariableName = "john"
```

- Snake style

```
my_variable_name = "john"
```

15. Constants in GO

- the variables must have a value that cannot be changed, you can use const
- the keyword const declares that a variable is constant which means it cannot be changed and is read-only

> Syntax

```
const constant_name type = value
```

16. Declaring a Constant

```
package main
import ("fmt")

const PI = 3.14

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

17. Rules of constants

- constant names follow the same rules as variable names
- constant names are normally written in uppercase letters
- constants can be declared inside and outside of a function

18. Types of constants

- defined constants
- undefined constants

Defined constants

- they are constants declared with a defined type

```
package main
import("fmt")

const A int = 1

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

Undefined constants

- undefined constants are constants that are not declared with a type

```
package main
import("fmt")

const A = 1
```

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

Unchangeable read-only constants

- when a constant is declared it is not possible to change its value afterwards

```
package main
import ("fmt")

func main () {
  const A = 1
  A = 2
  fmt.Println(A)
}
```

- result:

```
./prog.go:8:7: cannot assign to A
```

19. Declaration of several constants

- several constants can be grouped together to make everything more readable

```
package main
import("fmt")

const(
  A int = 1
  B = 3.14
  C = "hello"
)
func main(){
fmt.Println(A)
fmt.Println(B)
fmt.Println(C)
}
```

20. Output functions in GO

- there are three output functions in GO

  > Print()

> Println()

> Printf()

The Print() function

- the Print() function prints the arguments in their default way

```
package main
import ("fmt")

func main(){
  var a string = "hello world"
  fmt.Print(a)
}
```

- if we want to print the arguments in other lines we have to use \n

```
package main
import ("fmt")

func main(){
var i , j string = "hello" , "world"

fmt.Print(i, "\n")
fmt.Print(j, "\n")
}
```

> result

```
hello
world
```

- you can also use Print() for multiple variables

```
package main
import ("fmt")

func main() {
  var i,j string = "hello", "world"

  fmt.Print(i, "\n", j)
}
```

> result

```
hello
world
```

- for a space between the arguments use " " for example

```
fmt.Print(i, " ", j)
```

- Print also creates a space between the arguments if none is a string

```
package main
import ("fmt")

func main() {
var i,j = 10,20

fmt.Print(i,j)
}
```

> result

```
10 20
```

The Println function

- it is similar to Print() with the difference that a space and a blank line are generated between and after the arguments

```
package main
import ("fmt")

func main() {
var i,j string = "hello","world"

fmt.Println(i,j)
}
```

> result

```
hello world
```