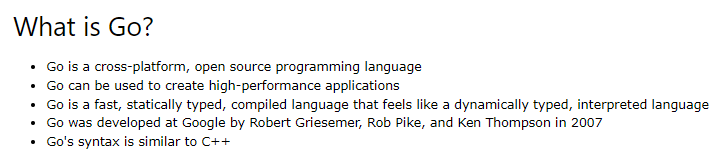
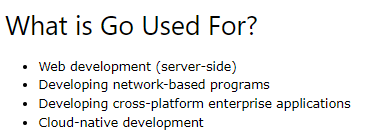
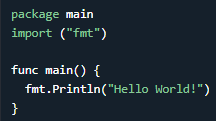
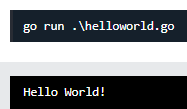
GO-LANG

Go is a popular programming language. Go is used to create computer programs.

Hello world in GO –



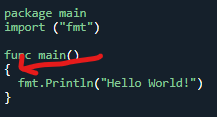
Go is statically types, fast run time, compiled, fast compiled, supports concurrency through goroutines and channels, does not support classes and objects also not support inheritance.



If you want to save the program as an executable, type and run: 

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. import ("fmt") lets us import files included in the fmt package. 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!".

Note –

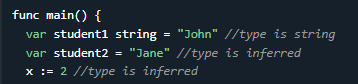
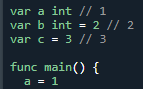
 In go, curly bracket { cannot come at the start of a line. Will give error



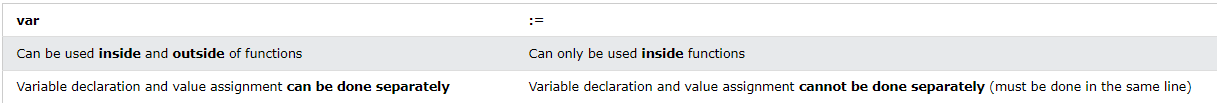
Comments - // or /\* \*/

Declaring (Creating) Variables - Use the var keyword, followed by variable name and type:  or With the := sign: 

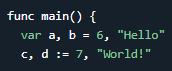
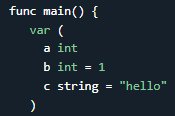
Note: In (:=) this case, the type of the variable is inferred from the value, It is not possible to declare a variable using :=, without assigning a value to it.

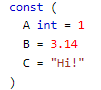
  

Variable Declaration Without Initial Value - In Go, all variables are initialized. So, if you declare a variable without an initial value, its value will be set to the default value of its type.



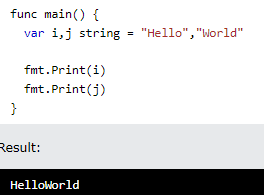
In Go, it is possible to declare multiple variables in the same line.  like this also –

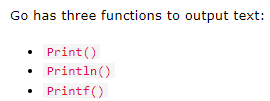
 or .

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

There are two types of constants: Typed constants & Untyped constants

 and untyped - > 

They are Unchangeable and Read-only

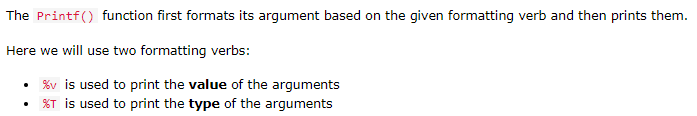


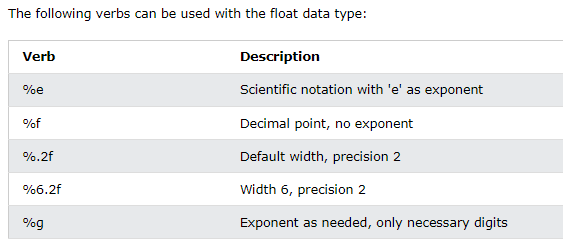
The Print() function prints its arguments with their default format.

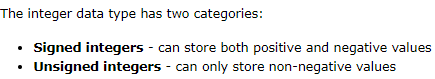
 or fmt.Print(j,” “ , i) -> with a space In between. mt.Print(i, "\n",j) alse

\n ->> creates a new line | Println()

* Print() inserts a space between the arguments if neither are strings:  // 10 20

 %% -> %

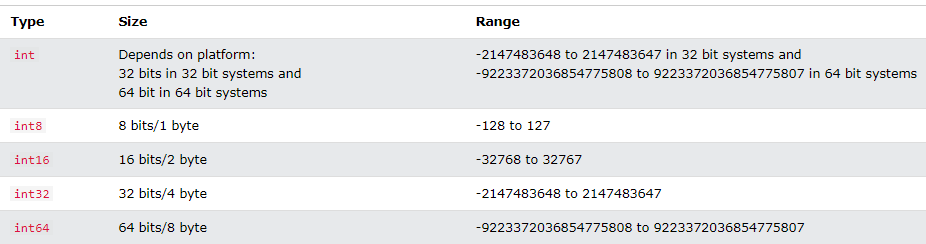
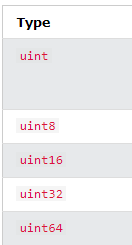


Go Data Types -> bool, int, float32, string

The default value of a boolean data type is false.

The default type for integer is int. If you do not specify a type, the type will be int.

Go has five keywords/types of signed integers:

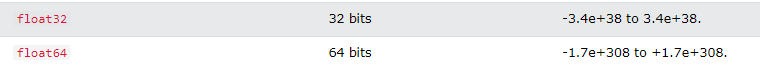
Unsigned integers, declared with one of the uint keywords, can only store non-negative values:

Go has five keywords/types of unsigned integers: uint,uint8,uint16,32,64

The type of integer to choose, depends on the value the variable has to store.

Eg - Result is an error because 1000 is out of range for int8 (which is from -128 to 127):

The float data types are used to store positive and negative numbers with a decimal point, like 35.3, -2.34, or 3597.34987. 2types -> float32, float64

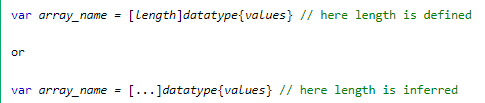
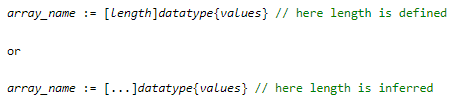


The string data type is used to store a sequence of characters (text). String values must be surrounded by double quotes.  no value initialised so empty string.

Reference to some popular formatting verbs - <https://www.w3schools.com/go/go_formatting_verbs.php>

Go Arrays

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

& with :=

Note: The length specifies the number of elements to store in the array. In Go, arrays have a fixed length. The length of the array is either defined by a number or is inferred (means that the compiler decides the length of the array, based on the number of values).



 -> access it the usual way arr[] | change it -> prices[2] = 50

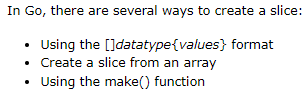
 -> all values 0, partial initialied will have those elements as stated rest all 0.

It is possible to initialize only specific elements in an array.

 -> 

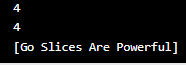
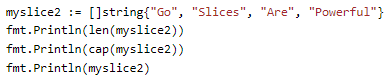
The len() function is used to find the length of an array.

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 two functions that can be used to return the length and capacity of a slice





Now how to create a slice from an array ->



End is exclusive

Create a Slice With The make() Function -> The make() function can also be used to create a slice.

 If the capacity parameter is not defined, it will be equal to length.

 Lets do more with slices

Just like array You can access a specific slice element by referring to the index number and can also change its value in a similar way.

You can append elements to the end of a slice using the append()function: 

 and 

Append One Slice To Another Slice – > myslice1 = append(myslice2) will make slice 1 and 2 equal



Note: The '...' after slice2 is necessary when appending the elements of one slice to another.

Unlike arrays, it is possible to change the length of a slice, by appending and slicing to increase or decrease the size of slice.

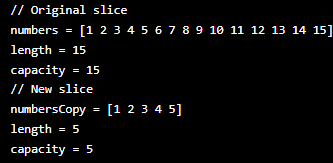
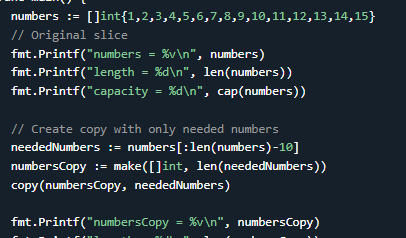
Memory Efficiency

When using slices, Go loads all the underlying elements into the memory. If the array is large and you need only a few elements, it is better to copy those elements using the copy() function.

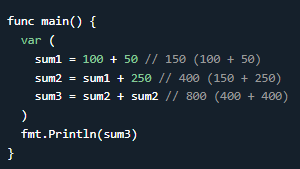
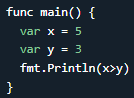
The copy() function creates a new underlying array with only the required elements for the slice. This will reduce the memory used for the program.

copy(dest, src)

The copy() function takes in two slices dest and src, and copies data from src to dest. It returns the number of elements copied.

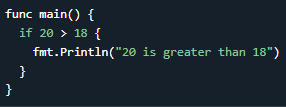


Go Operators – Although the + operator is often used to add together two values, it can also be used to add together a variable and a value, or a variable and another variable

  We already know about arithmetic operators and , assignment operator (=) to assign the value 10 to a variable called x. Also +=,-= , etc…

**Also >, < , >=, <=, ==, != ….. Logical -> ! , ||, && and then there is bitwise also.**

The if Statement -> Use the if statement to specify a block of Go code to be executed if a condition is true. Here there is no bracket around the condition.

 Also The brackets in the else statement should be like } else {:

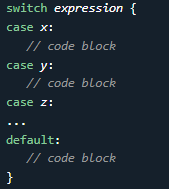
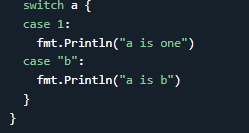
Having the else brackets in a different line will raise an error: 

Then there is if else if and else. Also The Nested if Statement

The switch Statement - 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.

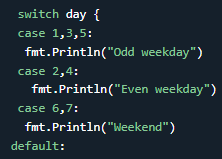
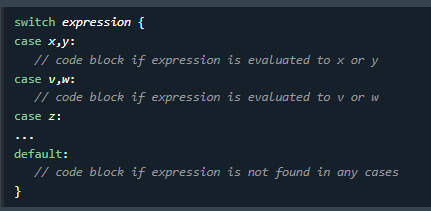
The default keyword specifies some code to run if there is no case match. No brackets around expression

All the case values should have the same type as the switch expression. Otherwise, the compiler will raise an error.

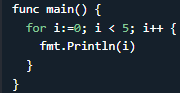
 Should be same type



The Multi-case switch Statement - It is possible to have multiple values for each case in the switch statement.



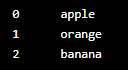
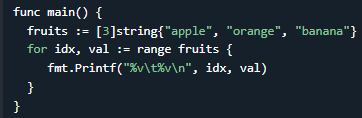
Go For Loops



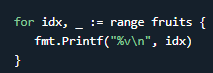
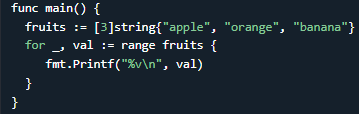
The continue statement is used to skip one or more iterations in the loop. It then continues with the next iteration in the loop. The break statement is used to break/terminate the loop execution. continue and break are usually used with conditions.

The Range Keyword ( This is new)

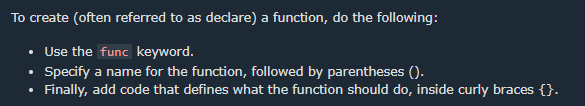
The range keyword is used to more easily iterate over an array, slice or map. It returns both the index and the value. The range keyword is used like this.



Tip: To only show the value or the index, you can omit the other output using an underscore (\_). Also if, we want to omit the values (idx stores the index, val stores the value):



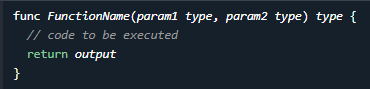
Go Functions - A function is a block of statements that can be used repeatedly in a program. A function will not execute automatically when a page loads. A function will be executed by a call to the function.

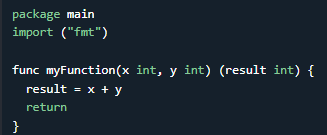


Function names are case-sensitive and A function name must start with a letter

Parameters and Arguments -> First name of parameter then type.

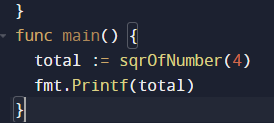
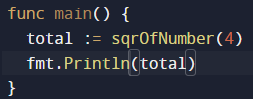


Return Values - If you want the function to return a value, you need to define the data type of the return value (such as int, string, etc), and also use the return keyword inside the function: Interestingly u can also do it like this –

and  for 1+2 So returning result is already specified in function declaration.

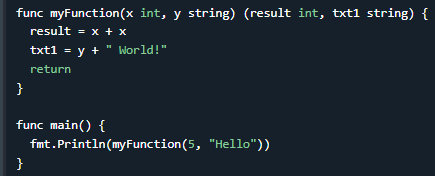
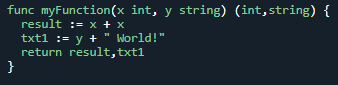
If you want function to return something then only you have to specify the return type otherwise only parameters and executes statement under.

Note –

 a) b)

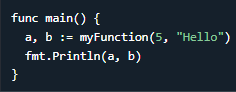
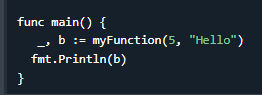
With this code You will get error as the total hold int and when we write Printf that means we are printing some string value. And when we write println that means its for line printing.( can be any value) so, Img b code works and print 16 as value. You can also store the return value in a variable, like this.

Go functions can also return multiple values. Here, myFunction() returns one integer (result) and one string (txt1):

 or 

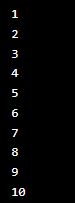
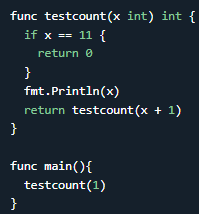


Here, we store the two return values into two variables (a and b):

a) b)

If we (for some reason) do not want to use some of the returned values, we can add an underscore (\_), to omit this value. Like in b) and if we want to omit the second returned value (txt1 - which is stored in variable b): 

Recursion Functions -> A function is recursive if it calls itself and reaches a stop condition.



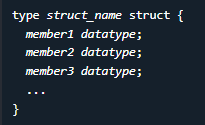
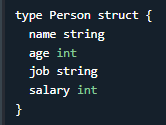
Recursion is a common mathematical and programming concept. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be careful with recursion functions as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

Go Struct

A struct (short for structure) is used to create a collection of members of different data types, into a single variable. While arrays are used to store multiple values of the same data type into a single variable, structs are used to store multiple values of different data types into a single variable. A struct can be useful for grouping data together to create records.

**To declare a structure in Go, use the type and struct keywords:**

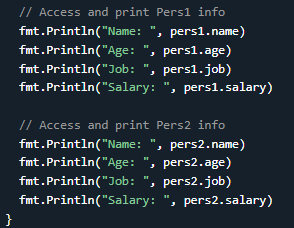
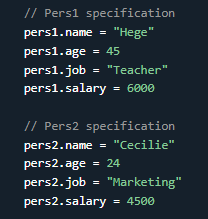
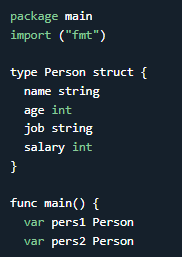
 -> 

Notice that the struct members above have different data types. name and job is of type string, while age and salary is of type int.

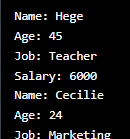
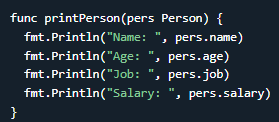
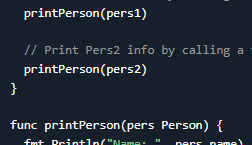
Access Struct Members

To access any member of a structure, use the dot operator (.) between the structure variable name and the structure member:

Whooh It seems cool , My bad Why I never cared about learning struct. They allow you to define custom data types by grouping together variables (fields) under a single type name. This can be very useful for organizing and managing complex data in your programs.



You can also pass a structure as a function argument, like this:

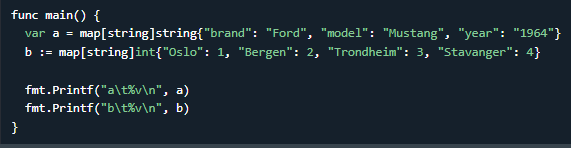


Go Maps

* Maps are used to store data values in key:value pairs.
* Each element in a map is a key:value pair.
* A map is an unordered and changeable collection that does not allow duplicates.
* The length of a map is the number of its elements. You can find it using the len() function.
* The default value of a map is nil.
* Maps hold references to an underlying hash table.
* Go has multiple ways for creating maps.

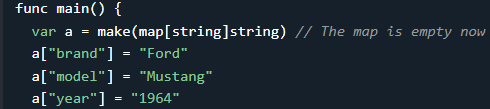
Create Maps Using var and :=







Create Maps Using make()Function: I feel it’s a lengthy way unnecessarily





Similarly It could be for int and string -> v:= make(map[int]string)) so …… v[2] = “Oggy”

There are two ways to create an empty map. One is by using the make()function and the other is by using the following syntax. 

Note: The make()function is the right way to create an empty map. If you make an empty map in a different way and write to it, it will causes a runtime panic.

The map key can be of any data type for which the equality operator (==) is defined. These include:

Booleans, Numbers, Strings, Arrays, Pointers, Structs Invalid key types are: Slices, Maps, Functions

The map values can be any type. Even functions.

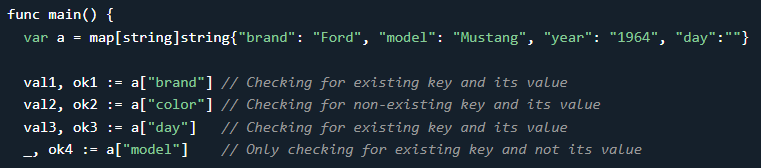
You can access map elements by:  eg - 

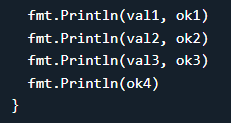
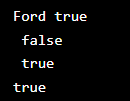
Updating or adding an elements are done by: 

Removing elements is done using the delete() function. 

You can check if a certain key exists in a map using:  Here the value – ok gives true or false to indicate that key exist or not, and value – val returns the value if the key exist. If not exist then ok will return just false. Val will not return anything in such case.

If you only want to check the existence of a certain key, you can use the blank identifier (\_) in place of val.

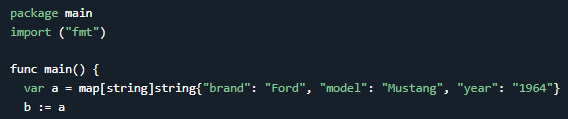


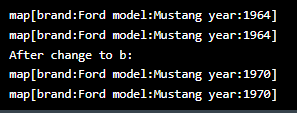
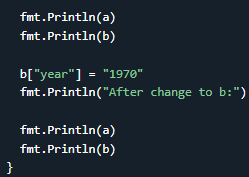
 

we checked for existence of different keys in the map. The key "color" does not exist in the map. So the value is an empty string (''). The ok2 variable is used to find out if the key exist or not. Because we would have got the same value if the value of the "color" key was empty. This is the case for val3.

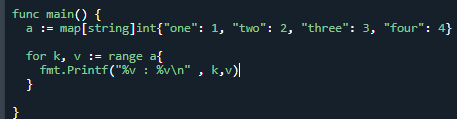
LET ME show you an interesting thing –

Maps are references to hash tables. If two map variables refer to the same hash table, changing the content of one variable affect the content of the other.





Iterate Over Maps -> You can use range to iterate over maps.

I am leaving the Iterate Over Maps in a Specific Order, It feels unnecessary!