When you use var to DECLARE a variable, the value of the variable DECLARED is set to the ZERO VALUE.

From the language spec:

"When storage is allocated for a variable, either through a declaration or a call of `new`, or when a new value is created, either through a composite literal or a call of `make`, and no explicit initialization is provided, the variable or value is given a default value. Each element of such a variable or value is set to the *zero value* for its type: `false` for booleans, `0` for integers, `0.0` for floats, `""` for strings, and `nil` for pointers, functions, interfaces, slices, channels, and maps."

REMEMBER THIS

declare + assign = initialize

DECLARE

You can declare a variable to be of a certain type.

var x int

I DECLARE that the variable x exists and is of type int

ASSIGN

You can assign a value to a variable.

x = 42

I am ASSIGNING the value 42, which is of type int, to the variable x, which has been declared to hold a value of type int.

When you just declare a variable "and no explicit initialization is provided, the variable or value is given a default value." The default value a variable is given when it is only declared is the ZERO VALUE.

Here are the ZERO VALUES for the different types: `false` for booleans, `0` for integers, `0.0` for floats, `""` for strings, and `nil` for pointers, functions, interfaces, slices, channels, and maps.

NIL for pointers, functions interfaces, slices, channels, and maps.

So when you only DECLARE you get the ZERO VALUE.

How do we declare?

var x int

var xi []int

var m map[string]int

What would be the zero values for those variables?

the zero value for x: 0

the zero value for xi: nil

the zero value for m: nil

The last word to understand here is ALLOCATED.

Here is my rewording of the golang spec paragraph we have been looking at: "When storage is ALLOCATED for a variable through a declaration, or when a new value is created through a composite literal or a call of `make`, and no explicit initialization (assignment of a value) is provided, the variable is ASSIGNED the zero value for its type."

So what is allocation?

When I DECLARE a variable, the "run time" (don't think run time like java, there's no jvm here, this is compiled - think "run time" in the sense of the Go SDK determining what happens, when) ... continuing ... When I DECLARE a variable, the "run time" has to ALLOCATE memory to store that variable's value - regardless of whether or not that value is 0, 0.0, "", false, or nil.

FYI - you saw `new` in the golang spec paragraph we have been studying (way at the top of this post). Do not use `new`. Using `new` is not recommended. It is preferred to use var or make.

Also note, nil is a predeclared identifier - just like the predeclared identifiers for types, constants, and functions:

Predeclared identifiers

The following identifiers are implicitly declared in the universe block:

```
Types:
       bool byte complex64 complex128 error float32 float64
       int int8 int16 int32 int64 rune string
       uint uint8 uint16 uint32 uint64 uintptr

Constants:
       true false iota

Zero value:
```

```
        nil

Functions:
        append cap close complex copy delete imag len
        make new panic print println real recover
```

https://golang.org/ref/spec#Predeclared_identifiers

I hope this helps!