

● JANUARY 2026 SERIES

FROM GO BUILD TO GO RUN

GOLANG 2026 - NIV RAVE

#03

VARIABLES, CONSTANTS, THE `:=` CONTROVERSY

WHEN TO USE SHORT DECLARATIONS.



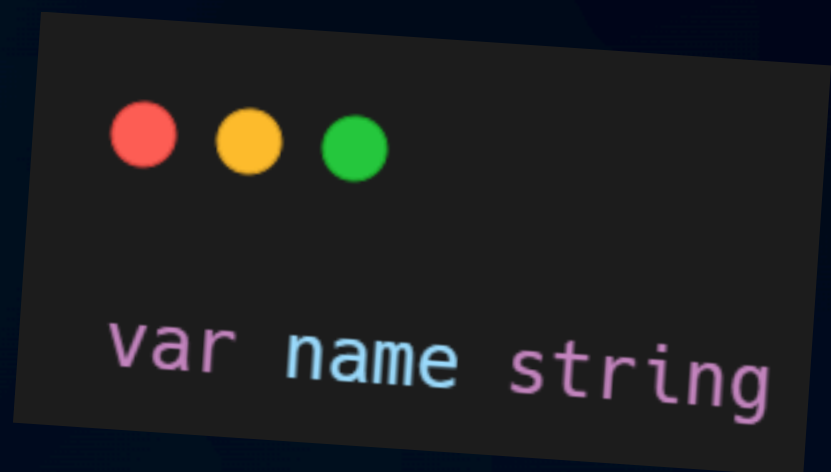


The Go Trinity

Go gives you options. Choosing the right one is about intent, not just saving keystrokes.



```
const maxRetries = 10
```



```
var name string
```



```
pi := 3.14
```



#1 - Short Declaration

The := Operator

Use this for **local** variables inside functions where the type is obvious



```
user := "niv_rave"  
count, active := 10, true
```

It's concise, but don't overdo it with long functions – it can make tracking types difficult for the next developer (or for the future you 😊)



#2 - The Case for "var"

Why we still need "var"?

```
package db

import "time"

var (
    databaseURL string
    timeout     time.Duration
    retries     int
)

func Connect(){
    var currentAttemp = 1
    ...
}
```

Use var when you aren't assigning a value immediately (initializing to "zero value")
or when you want to group package-level variables



#3 - Constants: Go's Guardrail

Constants are stricter here



```
const MaxConnections = 100  
const pi = 3.14  
const author string = "Niv"
```

In Go, constants are evaluated at compile-time. They aren't just "unchangeable variables", they are literal values the compiler relies on



Inference is Not "Dynamic"

Static Typing in Disguise

```
tmp := 6    // Translates to var tmp int  
tmp = 3.14  // Compiler error - cannot use 3.5 (untyped float constant) as int value in assignment
```

Just because you didn't write `int`, doesn't mean Go is dynamic like Python. Go is **statically typed**. The compiler infers the type once, and it never changes.





Shadowing: The Silent Bug

The Shadowing "Gotcha"

Watch the scope!

```
x := 10
if true {
    x := 6 // This is a NEW x, only inside this block!
    fmt.Println(x, " Inside block") // Prints 6
}
fmt.Println(x, "Outside block") // Prints 10
```



```
go run main.go
6  Inside block
10 Outside block
```

This is a common bug for newcomers. Using `:=` inside a block can accidentally "shadow" a variable from an outer scope.





The "Architecture" of Declarations

Intent-Based Declaration

Package Level (Global): You must use `var` or `const`. Short-variable declaration (`:=`) is not allowed here. This forces you to be explicit about global state.

Function Level (Local): Use `:=` for 90% of cases to keep code readable and concise.

The "Zero-Value" Pattern: Use `var` when you need a variable now but don't have the data yet.

TREAT YOUR PACKAGE-LEVEL DECLARATIONS AS THE "PUBLIC API" OF YOUR FILE. IF IT DOESN'T NEED TO BE THERE, MOVE IT INSIDE A FUNCTION USING `:=`





To summarize:

- `:=` is for local, "obvious" values.
- `var` is for zero-values, grouping, and package scope.
- `const` is for immutable, compile-time truths.

Are you a `:=` minimalist or a `var` traditionalist? Let's argue (politely) in the comments!

