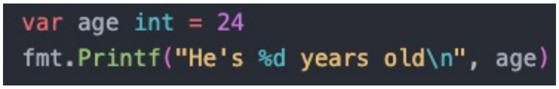
Section 4

(The fundomentals of Go)

14. Variables, zero values, blonk identifier

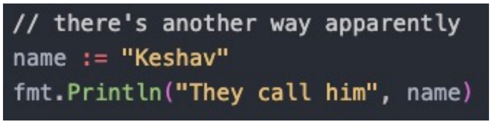
Variables

→ Here are some of the ways we can declare and assign values to a variable in Go



```go
var age int = 24
fmt.Printf("He's %d years old\n", age)
```

→ Explicitly specifying the type of value it will hold



```go
// there's another way apparently
name := "Keshav"
fmt.Println("They call him", name)
```

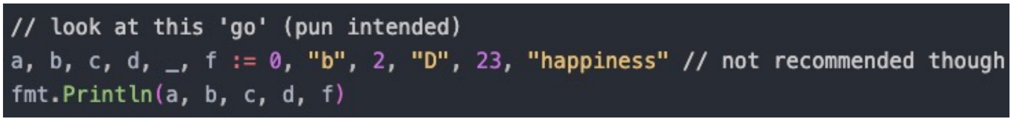
→ Compiler will check and dynamically set the type for the variable

Q Since compiler can set the type of the variable dynamically, does it mean that Go is 'dynamically typed' ?

→ No. It is statically typed language.

→ Even though the type is set dynamically, once the type for a variable is set, it later can not be changed !

```
age := 24
age = 50        //allowed
age = "6650"    // not allowed
```



```go
// look at this 'go' (pun intended)
a, b, c, d, _, f := 0, "b", 2, "D", 23, "happiness" // not recommended though
fmt.Println(a, b, c, d, f)
```

↳ We can initialize and declare multiple variables of different types within the same line

→ This approach is generally not recommended

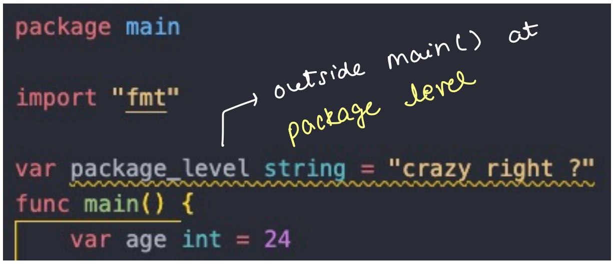
→ If you want to still, it's cleaner to declare and initialize multiple variables of the same type

Q How is a) var name = "Keshav" different from b) name := "Keshav" ?

→ b) is a shorthand way to declare and initialize variables

→ Scope of use → a) can be used at both the package level as well as inside functions

→ On the other hand b) can only be used at the function level

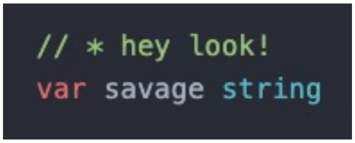


```go
package main

import "fmt"

var package_level string = "crazy right ?"
func main() {
    var age int = 24
```

→ Explicit Type Declaration → with var we get the added option of specifying the type explicitly if needed.

→ Declaration w/o Initialization → with var, we can declare a variable without immediately assigning it a value.

↳ NOTE → If you don't use your variable in the code afterwards, you will get a compile time error



```
// * hey look!
var savage string
```

→ 'immediate' initialization / usage not required

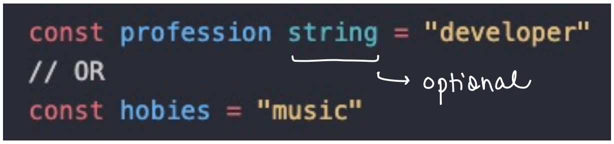
→ However with b) we would always require an initialization

Fixed Variables

Variables declared with the const keyword serve a different purpose

→ Immutable : Once declared, the value it stores can not be changed

→ const must be defined with values known at compile time itself

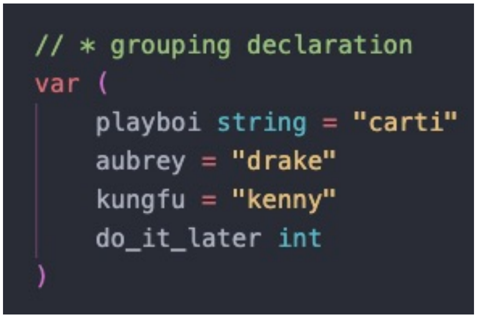


```go
const profession string = "developer"
// OR
const hobies = "music"
```

optional

→ Scope : Like var, they are package level

Grouped Declarations

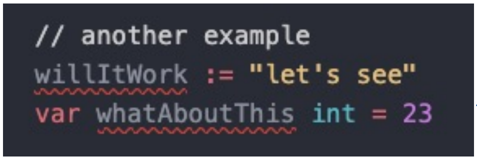


```go
// * grouping declaration
var (
    playboi string = "carti"
    aubrey = "drake"
    kungfu = "kenny"
    do_it_later int
)
```

→ Clear, more organized way to declare multiple variables

→ Same feature is available for const as well (remember to initialize it immediately though)

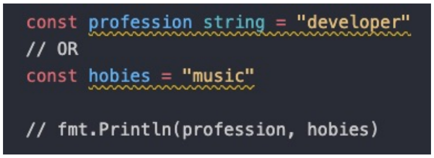
NOTE → Variables which have been declared but are not being used will throw compile time errors!



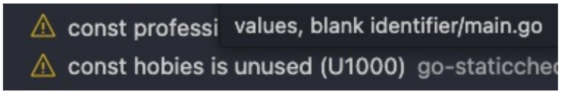
```go
// another example
willItWork := "let's see"
var whatAboutThis int = 23
```

Red squigglies

→ const variables will show warnings but will compile effectively



```go
const profession string = "developer"
// OR
const hobies = "music"

// fmt.Println(profession, hobies)
```



```
⚠ const professi values, blank identifier/main.go
⚠ const hobies is unused (U1000) go-staticchec
```

General Guideline

→ When we do this var <identifier> <type> and don't assign it a value, they get assigned the zero value

→ All the types have their own zero value

int, int8, int16 ...
↓
0

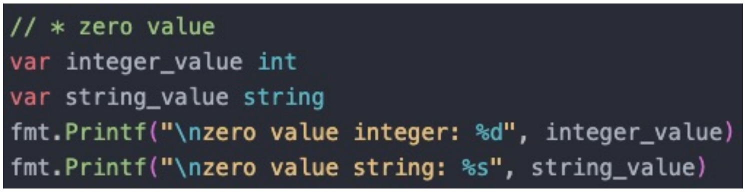
float32, float64
↓
0.0

bool
↓
false

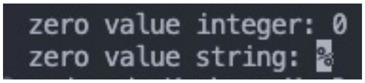
string
↓
""

... and so on

→ So, generally speaking use short declaration operator, if you want to get the zero value, use the specified syntax



```
// * zero value
var integer_value int
var string_value string
fmt.Printf("\nzero value integer: %d", integer_value)
fmt.Printf("\nzero value string: %s", string_value)
```



```
zero value integer: 0
zero value string: %
```