



# GOLANG

(RAFAEL ABREU DE CRISTO) && (JULIEN DOIRON)

# DECLARING VARIABLES

Canva

2 keywords

## var

- Declares a mutable variable
- Specify type or value (or both)



:=

- Same as declaring a var
- Only specify Value

## const

- Declares an immutable variable
- Must specify value



# Declaring Variable Examples

## var

```
var num1 = 3
```

```
var num2 int  
num2 = 4
```

```
var num3 int = 3
```

## :=

```
num4 := 4
```

## const

```
const NUM5 = 5  
const NUM6 int = 6
```



# DATA TYPES



4 main data types

## bool

- True or False

## int

- Whole number
- Unsigned and Signed
- Default bits depend on system (64 bits for 64 bit systems)

## float

- Decimal point numbers
- Default is float64 if not specified

## string

- stores characters
- Only double quotes



# Data Types Examples

## bool

```
var x bool = false  
var y bool = true
```

## float

```
var num1 float32 = 12.3  
var num2 float64 = 65.1234  
var num3 = 34.5 // defaults to float64
```

## string

```
var text string = "hello"
```



# Data Types Examples

## int

```
var num4 int8 = 127
var num5 int16 = 32767
var num6 int32 = 2147483647
var num7 int64 = 9223372036854775807

var num8 uint8 = 255
var num9 uint16 = 65535
var num10 uint32 = 4294967295
var num11 uint64 = 18446744073709551615

var num12 int = 9223372036854775807 // Defaults to int64
```



# ARRAYS

Canva

2 ways to declare (cannot be const)

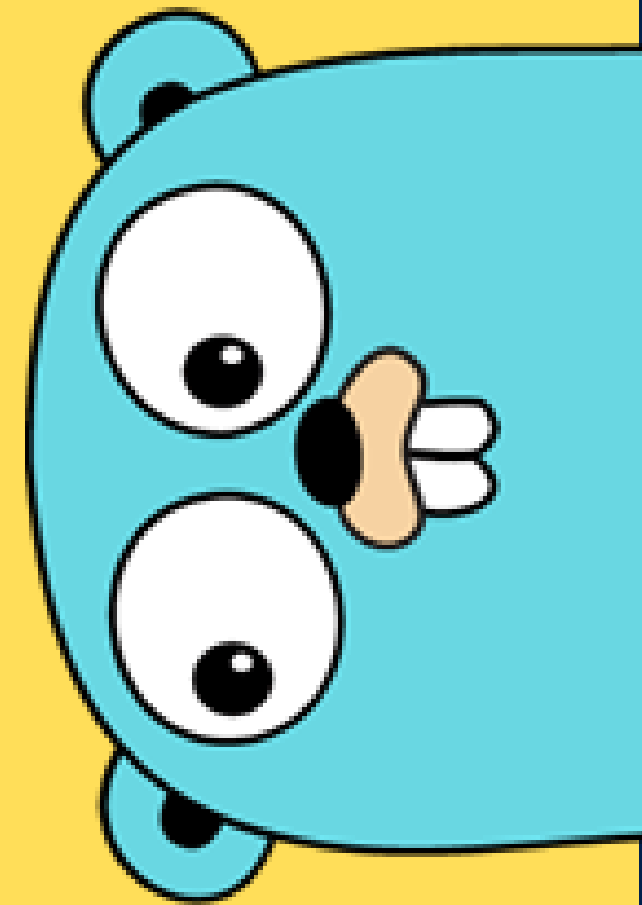
- `var array_name = [length]datatype{values}`
- `var array_name = [...]datatype{values}`

```
var names = [2]string{"Julien", "Rafael"}
```

```
var names = [...]string{"Julien", "Rafael"}
```

```
var names = [2]string{}  
names[0] = "Julien"  
names[1] = "Rafael"
```

function that works with arrays: `len()`



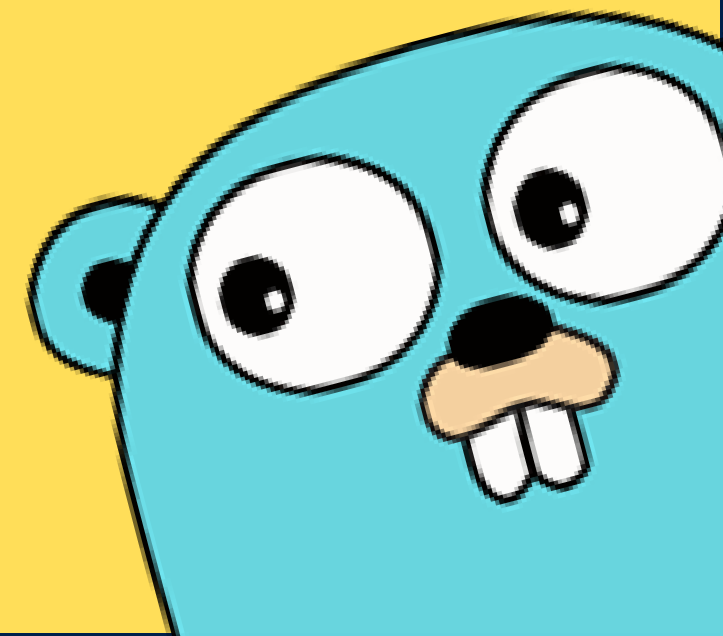
# SLICES



Like arrays but more flexible

- `var slice_name = []datatype{values}`
- `var slice_name = array_name[start: end?] // slice from array`
- `var slice_name = make([]datatype, length, capacity?)`

```
var slice1 = []int{3, 4, 5, 5} // [3,4,5,5]
var slice2 = []int{}           // []
var slice3 = make([]int, 5)    // [0,0,0,0,0]
var slice4 = names[0:]         // ["Julien", "Rafael"]
```





# Modifiying Slices

## append()

*slice\_name* = apend(*slice\_name*, *element1*, *element2*, ...)

```
var slice5 = []int{1, 2, 3}
var slice6 = append(slice5, 4, 5, 6) // [1,2,3,4,5,6]
```

```
var slice5 = []int{1, 2, 3}
var slice6 = []int{4, 5, 6}
var slice7 = append(slice5, slice6...) // [1,2,3,4,5,6]
```

spread operator



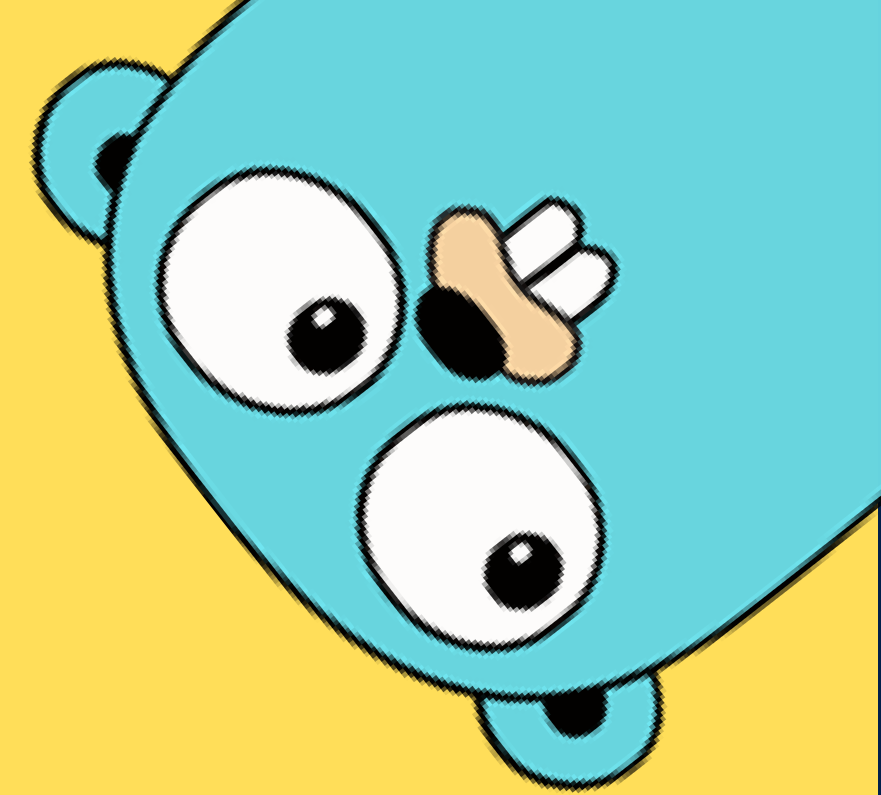
# IF AND SWITCH

Canva

```
if 10 > 9 {  
    fmt.Println("10 is greater")  
}
```

```
var number int = 1  
  
switch number {  
case 1:  
    fmt.Println("The number is 1")  
case 2:  
    fmt.Println("The number is 2")  
case 3:  
    fmt.Println("The number is 3")  
default:  
    fmt.Println("The number was none of the above")  
}
```

no break; needed



# LOOPS



Only for loops in go

- Simple For Loop

```
for i := 0; i < 10; i++ {  
    fmt.Println(i)  
}
```



# LOOPS

Canva

Continue and Break keyword

```
for i := 0; i < 10; i++ {  
    if i == 5 {  
        continue // Will skip the current iteration  
    }  
  
    fmt.Println(i)  
}
```



# LOOPS

Canva

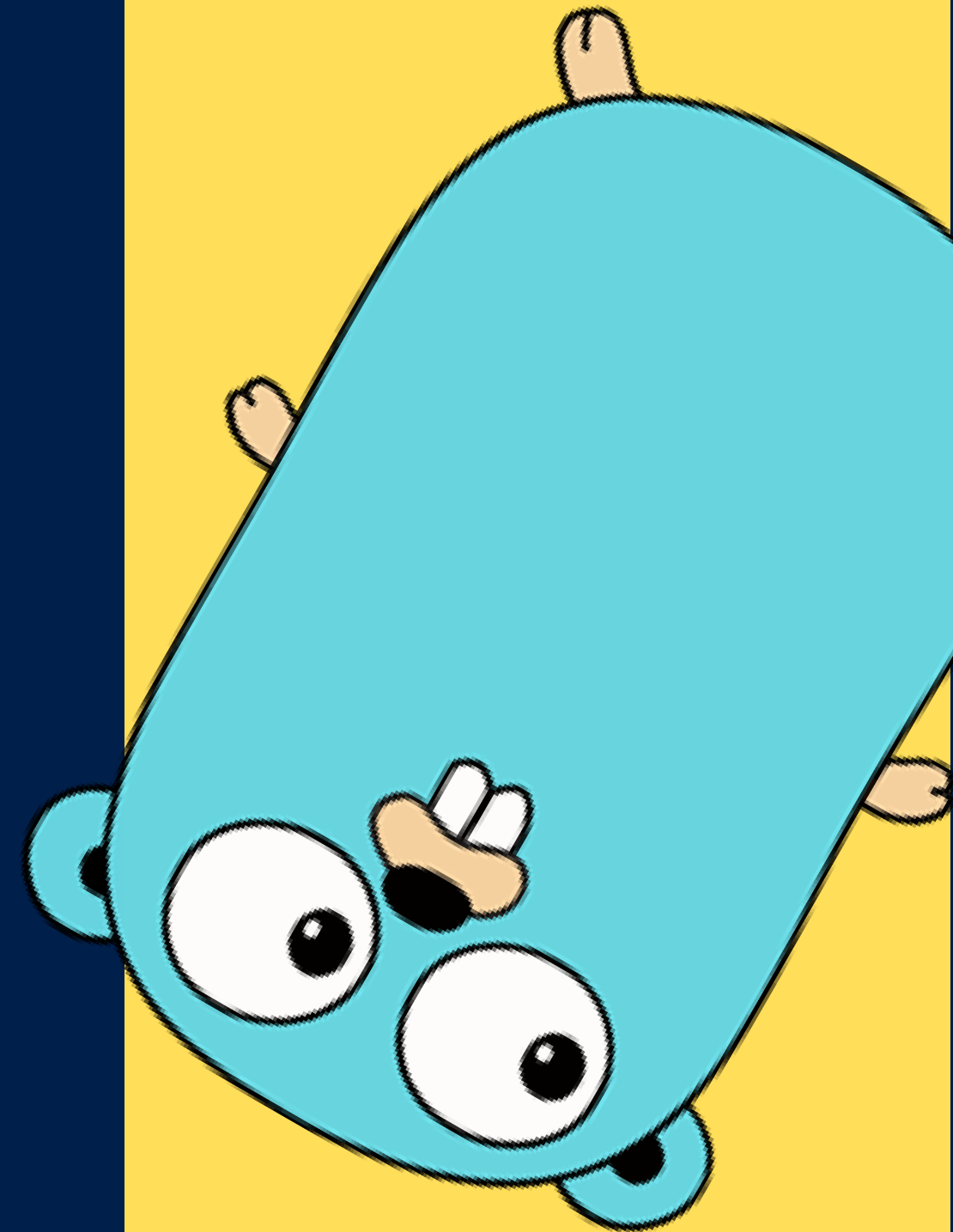
Range keyword

- for *index*, *value* := *array|slice|map*

```
var names = [3]string{"Julien", "Rafael", "Samuel"}

for idx, val := range names {
    fmt.Printf("index: %d value: %s\n", idx, val)
}
```

```
index: 0 value: Julien
index: 1 value: Rafael
index: 2 value: Samuel
```



# FUNCTIONS



func keyword

- `func function_name(param1 type, ...) type`

```
func addNumbers(num1 int, num2 int) int {  
    return num1 + num2  
}
```



# FUNCTIONS

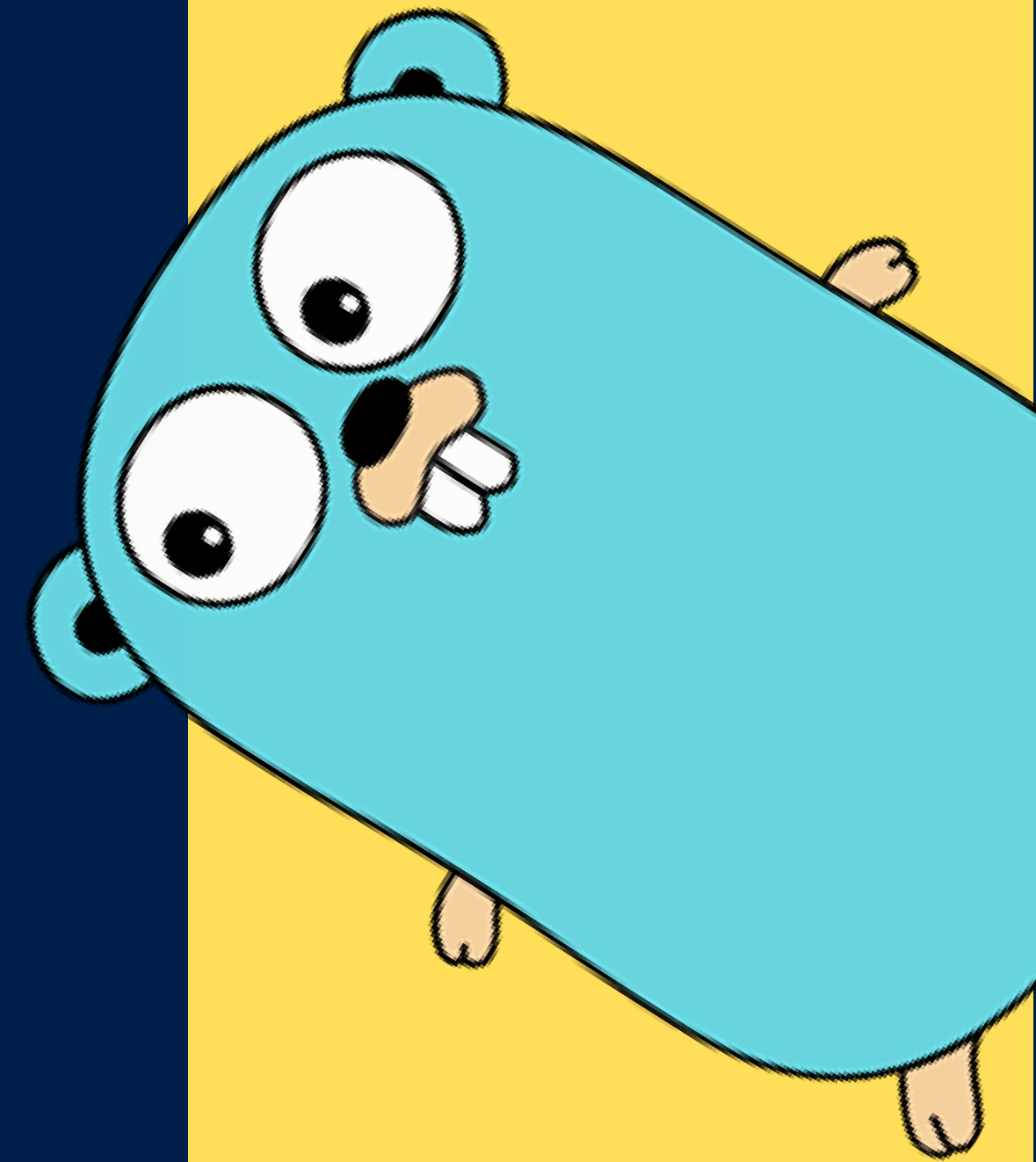


- `func function_name(param1 type, ...) (variable_name type)`

```
func addNumbers(num1 int, num2 int) (result int) {  
    result = num1 + num2  
    return  
}
```

- Add any number of return values

```
func addNumbers(num1 int, num2 int) (result1 int, result2 int) {  
    result1 = num1 + num2  
    result2 = num1 - num2  
    return  
}
```



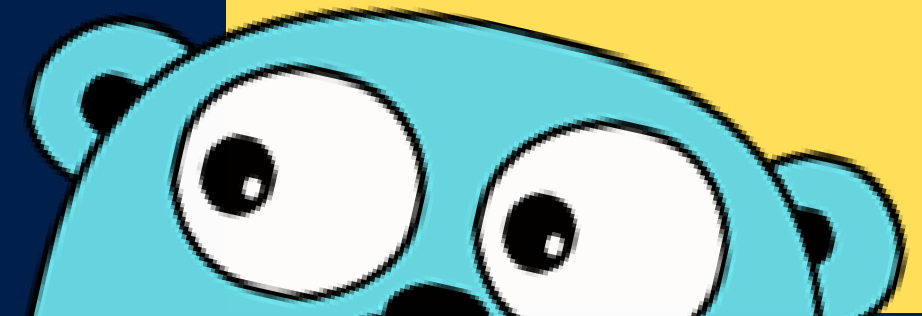
# STRUCT



Like a Class

- `type struct_name struct {`  
    *member1 datatype*  
    ...  
}

```
type Snowboard struct {  
    length int  
    style  string  
    brand  string  
    color  string  
}
```







# Struct

## Accessing properties

Access properties with dot operator

```
var snowboard1 Snowboard  
  
snowboard1.length = 154  
snowboard1.style = "Regular"  
snowboard1.brand = "Capix"  
snowboard1.color = "Brown"
```

## Printing properties

```
fmt.Println("Length: ", snowboard1.length)  
fmt.Println("Riding Style: ", snowboard1.style)  
fmt.Println("Brand: ", snowboard1.brand)  
fmt.Println("Color: ", snowboard1.color)
```

```
Length: 154  
Riding Style: Regular  
Brand: Capix  
Color: Brown
```

# STRUCT



Creating functions for structs

```
func (s Snowboard) printDetails() {  
    fmt.Println("Length: ", s.length)  
    fmt.Println("Riding Style: ", s.style)  
    fmt.Println("Brand: ", s.brand)  
    fmt.Println("Color: ", s.color)  
}
```



# STRUCT



Creating functions that will make changes to the object

Place star here to work with persisting data



```
func (s *Snowboard) addLength(length int) int {  
    s.length += length  
    return s.length  
}
```



This will make sure you are changing the values of the original object, and not a copy of it



**Thank  
you**

## References



- The Go Programming Language (n.d). Tutorial: Get started with Go  
<https://go.dev/doc/tutorial/getting-started>
- The Go Programming Language (n.d). Documentation  
<https://go.dev/doc/>
- The Go Programming Language (n.d). Effective Go  
[https://go.dev/doc/effective\\_go](https://go.dev/doc/effective_go)
- The Go Programming Language (n.d). Standard Library  
<https://pkg.go.dev/std>
- W3 Schools (n.d). Go Syntax  
[https://www.w3schools.com/go/go\\_syntax.php](https://www.w3schools.com/go/go_syntax.php)
- Youtube (n.d). Web application development with Buffalo, in Golang  
<https://www.youtube.com/watch?v=1mXWtP3EkLk>