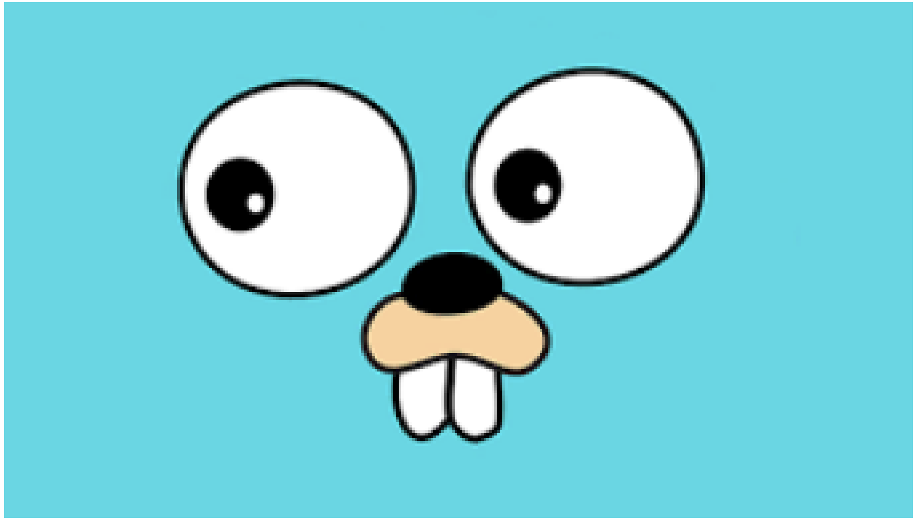
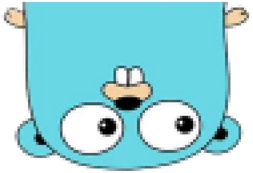


Phase 1: Introduction to Golang (Sessions 1-8)

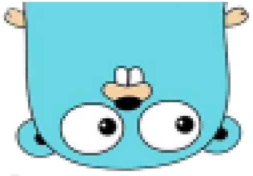
Objective: Build a strong foundation in Golang basics.

Session 5: Pointers and Memory Management



Discussion Points

- Understanding pointers in Go
- Pointers vs. values
- Using pointers with functions



Understanding pointers in Go

What is Pointer?

Pointers are a fundamental concept in computer science and are used in many programming languages, including Go. A pointer is a variable that stores the memory address data referenced by another variable. This allows you to manipulate the memory directly and create references to variables. A pointer variable contains the memory address of ***another*** value which means it points to that value in memory. The size of a pointer is **4 bytes** on 32-bit machines, and **8 bytes** on 64-bit machines, regardless of the size of the value they point to. These memory addresses are represented in **hexadecimal** values.

Declare a pointer `var ptr *type`

Assign memory address

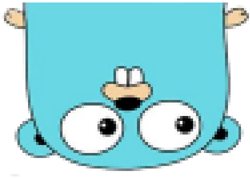
```
var num int = 5
var ptr *int

ptr = &num
```

prt

0xc00000a0d8 0xc000054050

*

num



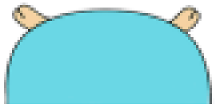
0xc00000a0d8

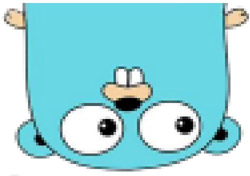
Understanding pointers in Go

What is Pointer?

Dereferencing a Pointer

```
func main() {
var num int = 5
var ptr *int

ptr = &num
fmt.Printf("*ptr points to num value = %d\n", *ptr)

// *ptr points to num value = 5
}
```

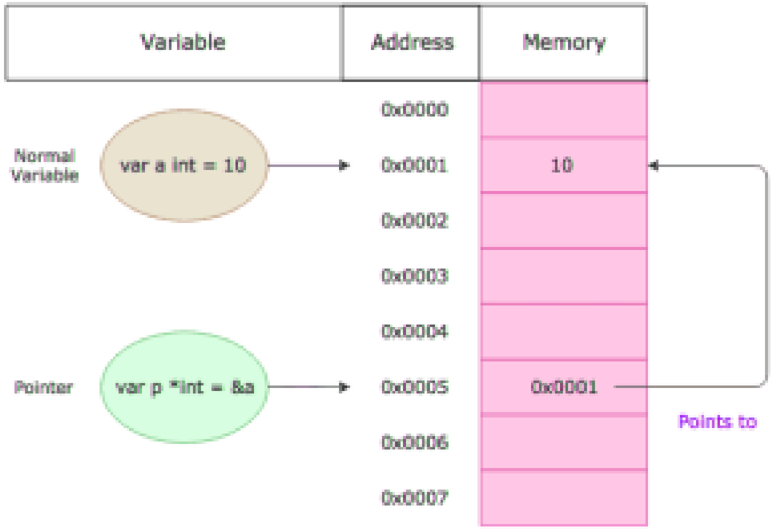




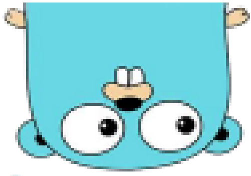
Understanding pointers in Go

What is Pointer?

Changing the variable value. The data at the memory location to which a pointer points can be both read and modified (written).



Variable		Address	Memory
		0x0000	
Normal Variable	var a int = 10	0x0001	10
		0x0002	
		0x0003	
		0x0004	
Pointer	var p *int = &a	0x0005	0x0001
		0x0006	
		0x0007	



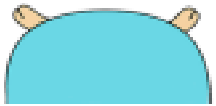
Understanding pointers in Go

What is Pointer?

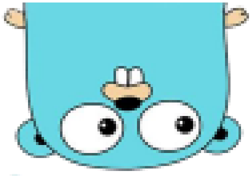
Nil Pointers in Golang and create pointers using **new()** value returned is a pointer to a newly allocated zero value of that type

```
func main() {
var p *int = nil // Making a nil pointer
*p = 0
}
// panic: runtime error: invalid memory address or nil pointer dereference
```

```
func main() {
var ptr = new(int)
fmt.Printf("*ptr points to num value = %d\n", *ptr)
}
```



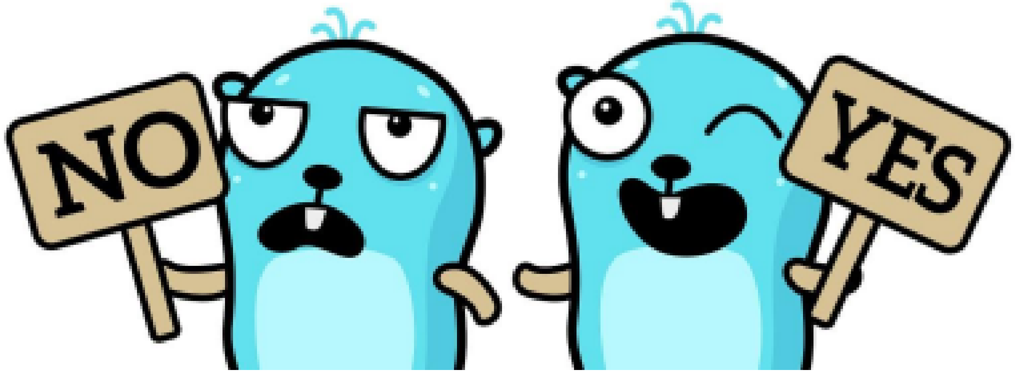
```
// *ptr points to num value = 0
```



Understanding pointers in Go

What about taking address of literal or constants?

You cannot take the address of a literal or a constant



Pointers vs. values

What is difference between Pointers and Values?

In Go, understanding the difference between pointers and values is essential, especially when considering performance, memory management, and how data is passed between functions.

Aspect	Values	Pointers
Data Passing	Passed by copy (each function gets its own copy of data)	Passed by reference (using the address of the original data)

Memory	Each variable/function has its own copy of the data.	Multiple variables/functions can reference the same data.
Performance	Copying large data structures like arrays can be expensive.	Passing pointers avoids copying large data structures, improving performance.
Modification of Data	Modifications to a value inside a function do not affect the original data.	Modifications through pointers affect the original data.
Usage Scenario	Use values when you don't need to modify the original data or performance isn't critical.	Use pointers when you want to modify the original data or avoid expensive copies.

Pointers vs. values

What is difference between Pointers and Values?

When to Use Values vs. Pointers

Use Values when:

- You do not need to modify the original value.
- The data type is small and copying it is inexpensive (e.g., integers, booleans, small structs). • Immutability is desired (i.e., ensuring functions don't accidentally change the data).

Use Pointers when:

- You need to modify the original value.
- The data type is huge

Using pointers with functions

What is Pointers with function?

In Go, all function parameters are passed by value by default. This means that a copy of the parameter value is passed to the function. However, we can use pointers to pass parameters by reference, which means that the function can modify the original parameter value.

We can also use pointers to return values from functions by reference, which allows the function to modify a variable outside its scope.

	Pass by Value	**Pass by Reference**
Function call	changeValue(p)	changeValue(&p)
Function receive & return	func changeValue(p int) (int, string)	func changeValue(p *int) (*int, *string)

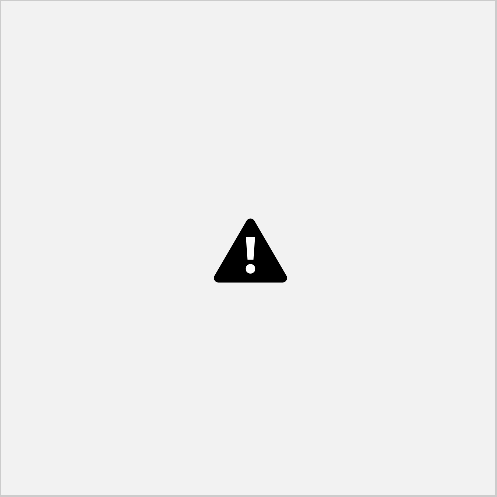
Using pointers with functions

What is Void Pointers?

Go supports void pointers, which are pointers that can point to **any data type**. Void pointers are often used in situations where the data type of pointer is unknown or needs to be determined at runtime. To declare a void pointer in Go, you can use the **unsafe.Pointer** type.

```
func main() {
   var ptr unsafe.Pointer

   num := 5
   ptr = unsafe.Pointer(&num)

   newInt := (*int)(ptr)
   fmt.Println(newInt) // 0xc00000a0d8
   fmt.Println(*newInt) // 5

   //fmt.Println(*ptr) //invalid operation: cannot indirect ptr (variable of type unsafe.Pointer)
}
```

Session Quiz



Join at

slido.com

# 1837155

Thank you.

Questions?

- Understanding pointers in Go
- Pointers vs. values

- Using pointers with functions

“Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning.”

- Albert Einstein