

Module 3

COMPOSITE TYPES AND COLLECTIONS



ARRAYS

- A fixed-size sequence of elements of a single type
- Size must be known at compile time
- Once declared, an array's size cannot change
- Syntax: `var a [5]int` (array of 5 integers)





SLICE

- **A dynamically-sized, flexible view into an array**
- **Always built on top of an underlying array**
- **Has three components: pointer to first element, length, and capacity.**
- **Syntax: `var s []int` (slice of integers)**

Maps

- Hash table implementation that stores key-value pairs.
- Keys must be comparable (can use == operator)
- Unordered collection
- Syntax: `var m map[KeyType]ValueType`



STRUCTS

- User-defined type that groups related data fields together.
- Each field has a name and a type

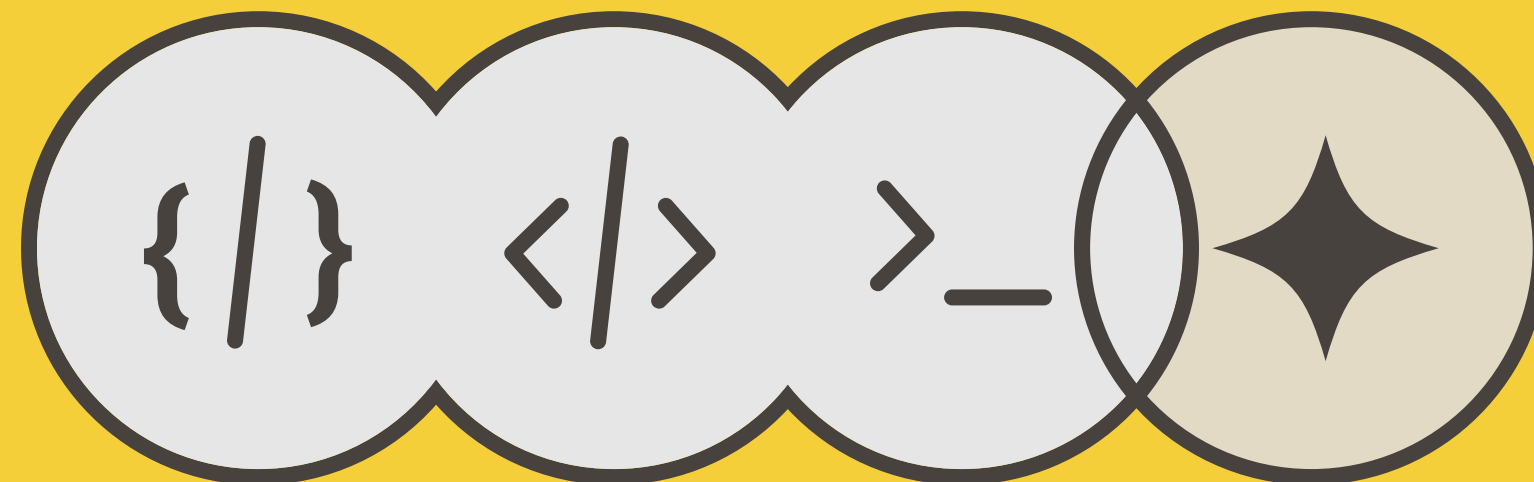
Syntax:

```
type Person struct {  
    Name string  
    Age  int  
}
```



Pointers

- Variable that stores the memory address of another variable.
- Zero value is nil
- Syntax: `var p *int` (pointer to an integer)



WHAT IS THE KEY DIFFERENCE BETWEEN AN ARRAY AND A SLICE IN GO?

- A) ARRAYS HAVE A FIXED SIZE, WHILE SLICES ARE DYNAMICALLY SIZED**
- B) SLICES CAN ONLY HOLD INTEGERS, WHEREAS ARRAYS CAN HOLD ANY TYPE**
- C) ARRAYS ARE ALWAYS PASSED BY REFERENCE, WHILE SLICES ARE PASSED BY VALUE**
- D) SLICES ARE STORED IN A DIFFERENT MEMORY REGION THAN ARRAYS**



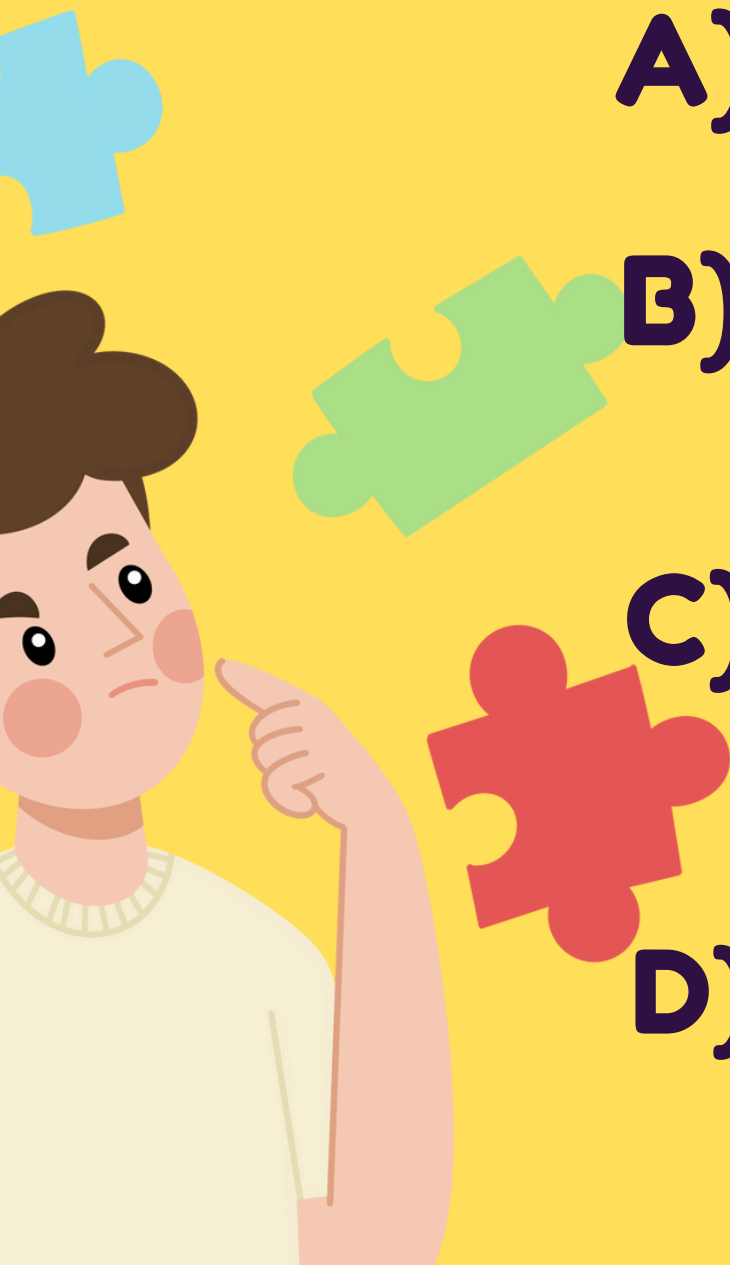
WHAT WILL HAPPEN WHEN PASSING A STRUCT BY VALUE INSTEAD OF BY POINTER?

A) THE ORIGINAL STRUCT WILL BE MODIFIED

B) THE FUNCTION WILL FAIL WITH A RUNTIME ERROR

C) A COPY OF THE STRUCT WILL BE CREATED, AND MODIFICATIONS WON'T AFFECT THE ORIGINAL

D) GO AUTOMATICALLY CONVERTS VALUE TYPES TO POINTERS WHEN NEEDED



THANK YOU

