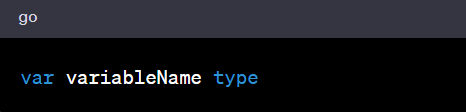
# Variables

%T type of variables

var variableName type



## **Default Values:**

variables that are declared but not explicitly initialized with a value are given a default value based on their type. Here are the default values for some common types:

* **Numeric Types (integers and floats):** The default value is **0**.
* **Boolean Type:** The default value is **false**.
* **String Type:** The default value is an empty string (**""**).
* **Pointers:** The default value is **nil**.
* **Arrays and Structs:** Each element or field within the array or struct is assigned a zero value.
* **Slices, Maps, and Channels:** The default value is **nil**.
* **Interfaces and Functions:** The default value is also **nil**.

Interfaces and Functions: The default value is also nil.

## **visibility of identifiers:**

visibility of identifiers (such as variables, functions, constants, and types) is determined by capitalization. The language uses a simple rule:

1. **Public**: Identifiers that start with an uppercase letter are exported and therefore visible from outside the package they are defined in. These identifiers are considered "public" and can be accessed by code in other packages.
2. **Private**: Identifiers that start with a lowercase letter are unexported and can only be accessed within the same package. These identifiers are considered "private" and are not visible or accessible from outside the package.

# Pointers In Go:

    var one int

    var ptr \*int

    one = 2

    ptr=&one;

    fmt.Println(\*ptr,one)

    var myptr=&one;

    \*myptr++

    fmt.Println(\*myptr)

# **Arrays in Golang:**

    var fruitList [4] string;

    fmt.Println("FruitList is ", fruitList)

    var fruitList =[4]string{"Apple","Mango","Peach","Avacado"}

## **Iterating Over an Array:**

    for index, value := range fruitList {

        fmt.Printf("Index: %d, Value: %s\n", index, value)

    }

# **Slices**

Slices are a more versatile and dynamic alternative to arrays in the Go programming language. Unlike arrays, slices can have a variable length and are more flexible when it comes to managing collections of elements.

    var myslice = []int{1,2,3,4,5}

    myslice = append(myslice, 6,7,8)

# Maps in Go

    var myMap map[string]int

    myMap = make(map[string]int)

    myMap["Ajay"] = 30

    myMap["Vijay"] = 20

    for key, values := range myMap {

        fmt.Printf("Key %v values %v", key, values)

    }

# Structs in Golang

type Person struct {

    Name string

    Age  int

}

func (p \*Person) SayHello() {

    fmt.Printf("Hello, my name is %s and I am %d years old.\n", p.Name, p.Age)

}

func (p \*Person) SetName(newName string) {

    p.Name = newName

}

func (p \*Person) SetAge(newAge int) {

    p.Age = newAge

}

func main() {

    person := Person{Name: "John", Age: 30}

    person.SayHello()

    person.SetName("Alice")

    person.SetAge(25)

    person.SayHello()

}