**What is Go (Golang)?**

Go (also known as **Golang**) is an open-source, statically typed, compiled programming language designed by **Google**. It was created in **2007** by **Robert Griesemer, Rob Pike, and Ken Thompson** and was officially released in **2009**.

**Key Features of Go**

✅ **Simple & Easy to Learn** – Go has a clean and minimal syntax, making it beginner-friendly.  
✅ **Fast Execution** – It is a compiled language, which makes it faster than interpreted languages like Python.  
✅ **Garbage Collection** – It automatically manages memory, reducing the risk of memory leaks.  
✅ **Concurrency Support** – Goroutines allow Go to handle multiple tasks simultaneously, making it ideal for high-performance applications.  
✅ **Cross-Platform** – Go runs on various operating systems, including Linux, Windows, and macOS.  
✅ **Strong Standard Library** – Go has a rich set of built-in packages for networking, file handling, and web development.

**Why Use Go?**

* 🚀 **Ideal for Backend Development** – Many cloud-native applications and APIs are built using Go.
* ⚡ **Efficient Concurrency** – Go is widely used for distributed systems and microservices.
* 📡 **Networking & Cloud Computing** – Many modern cloud services (e.g., Kubernetes, Docker) are written in Go.
* 🔥 **High Performance** – It competes with C and C++ in terms of speed but is easier to use.

**Hello, World! in Go**

Here’s a simple Go program to print "Hello, World!":

package main

import "fmt"

func main() {

fmt.Println("Hello, World!")

}

**How to Run?**

1. Install Go.
2. Save the code in a file **hello.go**.
3. Open a terminal and run:

go run hello.go

1. Output:

Hello, World!

**Lexer in Go (Golang)**

A **lexer** (or lexical analyzer) is the first step of a compiler. It takes raw source code as input and converts it into a sequence of **tokens** (such as keywords, identifiers, numbers, and operators).

In **Go**, the lexer is part of the Go compiler (go/parser and go/scanner), and it has some unique behaviors that many other programming languages **do not support**.

**What Makes Go’s Lexer Unique?**

1. **Automatic Semicolon Insertion**
   * Unlike languages like C, Java, or Python, **Go automatically inserts semicolons (;)** at the end of statements.
   * This allows Go to have a cleaner syntax without explicit semicolons.
2. **Line-Based Tokenization**
   * The lexer processes Go code **line by line**.
   * If a line ends in a valid expression (e.g., identifier, number, or keyword like return), Go **automatically** adds a semicolon.
3. **Simplicity in Parsing**
   * Many languages require complex parsing rules, but Go’s lexer enforces **strict syntax rules**, making it easier to analyze.

**Example of Semicolon Insertion in Go**

**✅ Valid Go Code Without Semicolons**

package main

import "fmt"

func main() {

fmt.Println("Hello") // No semicolomn needed

fmt.Println("World") // Go adds ‘;’ automatically

}

**Go Basics 🚀**

**1. Variables and Constants in Go**

**Declaring Variables**

In Go, you can declare variables using the var keyword or shorthand := named as **Walrus Operator**.

package main

import "fmt"

func main() {

// Using var (explicit type)

var name string = "Aman"

var age int = 23

// Using shorthand (implicit type inference)

city := "Karachi"

temperature := 30.5 // Go automatically infers float64

fmt.Println(name, age, city, temperature)

}

**Declaring Multiple Variables**

var a, b, c int = 1, 2, 3

x, y, z := 4.5, "Hello", true

**Constants in Go**

Constants are declared using the const keyword and **cannot be changed**.

package main

import "fmt"

func main() {

const pi = 3.1416

const country string = "Pakistan"

fmt.Println(pi, country)

}

❌ **Constants cannot use :=** (shorthand assignment is only for variables).

**2. Data Types in Go**

Go has four main data types:

| **Data Type** | **Description** | **Example** |
| --- | --- | --- |
| **int** | Integer values | var x int = 10 |
| **float64** | Decimal numbers | var y float64 = 5.5 |
| **string** | Text values | var name string = "Aman" |
| **bool** | Boolean (true/false) | var isGoEasy bool = true |

**Example**

package main

import "fmt"

func main() {

var age int = 25

var height float64 = 5.9

var name string = "GoLang"

var isFun bool = true

fmt.Println(age, height, name, isFun)

}

**3. Type Inference in Go**

Go automatically **infers** types based on the assigned value, making code more concise.

package main

import "fmt"

func main() {

x := 10 // Go infers it as int

y := 20.5 // Go infers it as float64

name := "Go" // Go infers it as string

isCool := true // Go infers it as bool

fmt.Printf("x is %T, y is %T, name is %T, isCool is %T\n", x, y, name, isCool)

}

**4. Operators in Go**

**Assignment Operators**

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| = | Assign value | x = 10 |
| += | Add and assign | x += 5 (same as x = x + 5) |
| -= | Subtract and assign | x -= 5 |
| \*= | Multiply and assign | x \*= 5 |
| /= | Divide and assign | x /= 5 |
| %= | Modulus and assign | x %= 5 |

🔹 **Example:**

package main

import "fmt"

func main() {

x := 10

x += 5

fmt.Println(x) // 15

x \*= 2

fmt.Println(x) // 30

}

**🔹 Summary**

✅ **Variables** use var or := for declaration.  
✅ **Constants** use const and cannot change.  
✅ **Data Types** include int, float64, string, and bool.  
✅ **Type Inference** allows Go to detect variable types automatically.  
✅ **Operators** include arithmetic, comparison, logical, assignment, and bitwise operations.

### ****Why Do Pointers Exist?****

Pointers exist to provide **direct memory access** and **efficient data manipulation**. They allow us to work with memory addresses instead of actual values, leading to better performance, flexibility, and lower memory usage in some cases.

#### **✅ Key Reasons for Using Pointers:**

* **Efficient Memory Usage** – Avoids unnecessary copying of large data structures.
* **Pass by Reference** – Allows functions to modify original variables.
* **Dynamic Memory Allocation** – Essential for data structures like linked lists, trees, etc.
* **Avoid Large Data Copying** – Reduces memory overhead when passing structs.
* **Low-Level System Interaction** – Used in system programming, networking, and hardware interfacing.

#### **⚠️ When NOT to Use Pointers:**

* For small variables (int, bool), as pointers add complexity.
* Risk of **nil pointer dereferencing** if not handled properly.

Would you like a quick code example? 🚀

FOR CREATING THE THE FILE OS IS USED

FOR READING AND OTHER MANIPULATION IOUTIL IS USED

Go mod tidy is an expensive cmd

Not only tidy   
after go mod all the operations are expensive