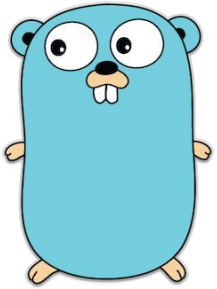

The Go Programming Language

An Introduction

Exploring Today's Topic

- ➔ Intro
- ➔ Project Setup
- ➔ Data types
- ➔ Functions and Control Structures
- ➔ Pointers, defer
- ➔ methods

Introduction

Go is a statically-typed, compiled programming language designed for simplicity and efficiency.

Developed by Google

Github Stats

- 115k stars
- 3.5k watching
- 17.3k forks

Go used by internet giants

Project Setup

To initialize the go project , CD into the project folder

Now enter : **go mod init <project name>**

Every go project need a main package with an entry function named **main**, which we define in the primary file.

```
package main
import "fmt"
func main() {
        fmt.Println("Hello World!")
}
```

For executing the code use,

```
go run <file name>
```

Datatypes & Initial Values

bool boolean string int int8 int16 int32 int64 uint uint8 uint16 uint32 uint64 byte // alias for uint8 rune // alias for int32 float float32 float64 complex complex64 complex128	• 0 for numeric types,float • false for the boolean type • "" (the empty string) for strings.

Arrays and slice

Array :

```
primes := [5]int {2, 3, 5, 7, 11}
```

Slice :

```
Sub := primes[1:3]
```

Maps

```
ages := make(map[string]int)
ages["Alice"] = 25
ages := map[string]int{
    "peter": 29,
    "manoj":40,
    "sita":25,
}
```

Control Structures

- **IF -ELSE**
- **FOR loop**
- **SWITCH**
- **DEFER**

Structs

```
type  Employee struct{
      Name string
      Id int
      Email string
      Department string
      }
   u1:=
Employee{"Alice",2006,"alice@gmail.com","
Fabric"}
   fmt.Println("Employee Details",u1)
   fmt.Println("Employee Name",u1.Name)
```

OUTPUT:

Struct in Golang

Employee Details {Alice 2006

alice@gmail.com Fabric}

Employee Name Alice

Pointer

Eg:

```
var p *int

i := 42

p = &i

fmt.Println(*p)

*p = 21
```

Functions

```
func add(a, b int) int {
    return a + b
}
 result := add(5, 3)
 fmt.Println("Result of addition:", result)
```

Methods

Functions

```
type Vertex struct {
        a1, b1 int
}
func add(v Vertex) int{
        return(v.a1 + v.b1)
}
func main() {
        v := Vertex{20,50}
        fmt.Println(add(v))
}
```

Methods

```
type Vertex struct {
        a1, b1 int
}
func (v Vertex) add() int {
        return(v.a1 + v.b1)
}
func main() {
        v1 := Vertex{20,50}
        fmt.Println(v1.add())
}
```

Importing from different packages

- Keep the first alphabet capitalized
- Different packages in different folders

JSON

JSON: JavaScript Object Notation, a lightweight, human-readable format for data exchange.

Encoding: Transforming Go data structures (structs, maps, slices, etc.) into JSON strings.

Decoding: Converting JSON strings back into Go data structures.

Interfaces

- Interfaces define what a type must do (methods), not how it does it (implementation).
- Think of them as blueprints outlining functionalities, not concrete structures.

Thank You :)