

Introduction to Go

First step into the Golang world

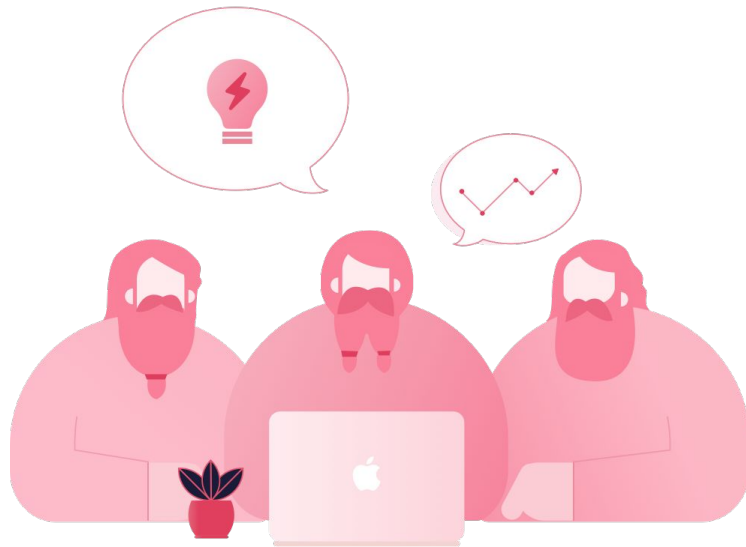


Author Name

Hieu Phan

@hieuphq

andy@d.foundation



Agenda

Introduction to Golang
development

1. Why Go?
2. Setting up Go Development
3. Go Basic
4. Q&A

Why Go?

Highlight the advantages of using Go as a programming language

Why Go?

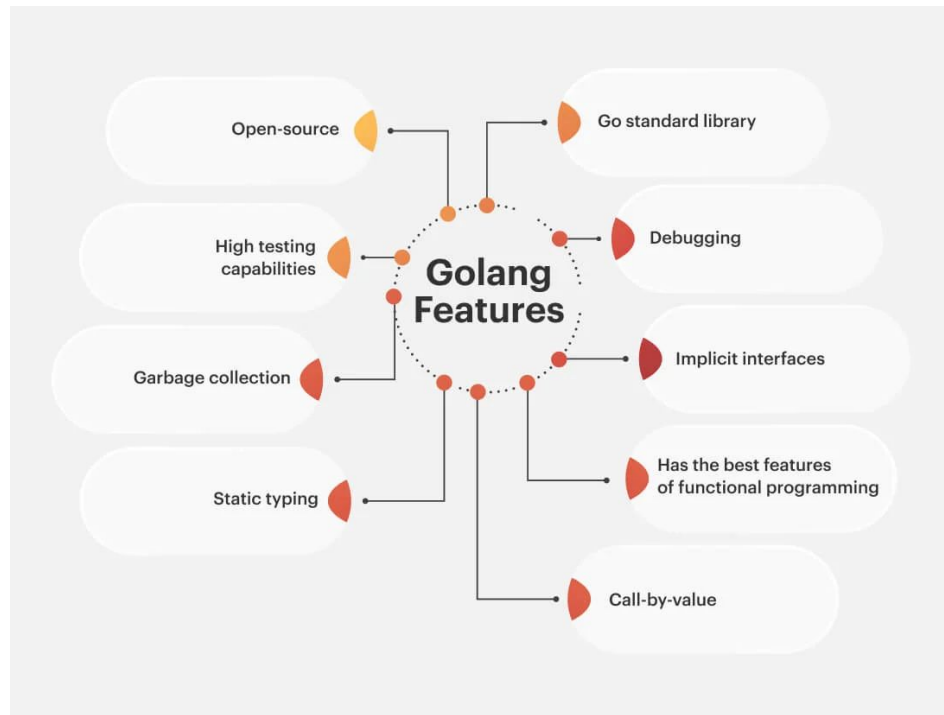
- Simplicity
- Performance
- Strong support for Concurrency



Go

Why Go?

- Built-in tooling
- Cross-platform compatibility
- Strong standard library

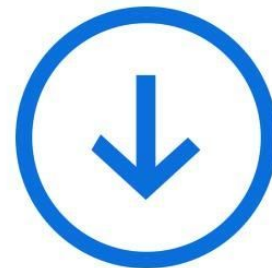


Setting up Go Development

Explain the steps to install and configure the Go development environment

Downloading and Installing Go

- Go to the official Go website (<https://go.dev/>) and navigate to the "Downloads" section.
- Choose the appropriate installer package for your operating system (Windows, macOS, or Linux).
- Run the installer and follow the on-screen instructions to install Go on your machine.



Setting up Environment Variables

- Go uses two important environment variables: GOROOT and GOPATH.
- GOROOT specifies the location where Go is installed on your system. In most cases, this variable is set automatically during the installation process.
- GOPATH specifies the location of your Go workspace, where you will store your Go projects and their dependencies.



Integrated Development Environments (IDEs)

Popular IDEs for Go include Visual Studio Code (with the Go extension), GoLand, and Atom (with the go-plus package). Install your preferred IDE and configure it to work with Go.

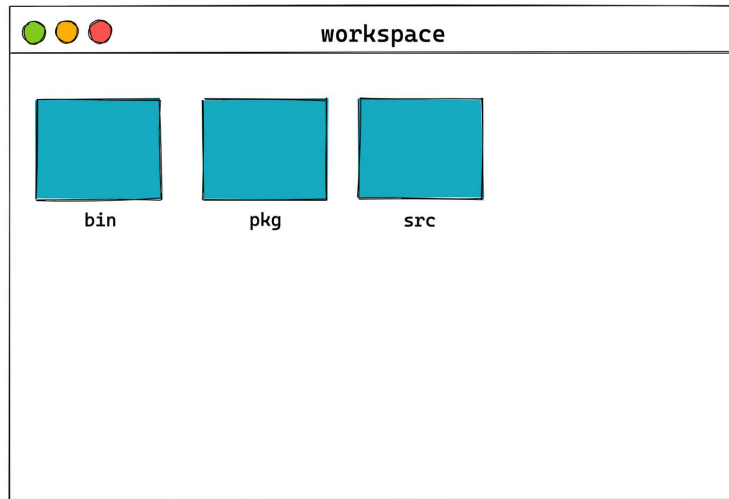


Go Basic

Writing First Go Program

Go Workspace Structure

- The src directory is where all Go source code files reside.
- The bin directory stores executable binaries generated by the Go compiler.
- The pkg directory contains compiled object files (.a files) generated during the build process.



Go Workspace Structure

- Let's say you're developing a Go package named `example.com/myapp`.
- Inside the workspace's `src` directory, you would create the following directory structure: `src/example.com/myapp`.
- All Go files belonging to the `myapp` package would reside within the `myapp` directory.

```
<GOROOT>/  
├── bin  
├── pkg  
└── src/  
    └── example.com/  
        └── myapp
```

Writing a "Hello, World!" Go API Server

1. Init project using go module
2. Create main.go file
3. Run project go run main.go
4. Invoke the server

```
1 package main
2
3 import (
4     "fmt"
5     "net/http"
6 )
7
8 func helloHandler(w http.ResponseWriter, r *http.Request) {
9     fmt.Fprint(w, "Hello, World!")
10 }
11
12 func main() {
13     http.HandleFunc("/", helloHandler)
14     http.ListenAndServe(":8080", nil)
15 }
16
```

Writing a "Hello, World!" Go API Server

curl http://localhost:8080/

```
● ~/w/g/d/hello >>> curl http://localhost:8080/  
Hello, World!  
○ ~/w/g/d/hello >>> 
```

Variables in Go

Declare a variable and specify its data type explicitly.

Declaration using the `:=` operator, where the data type is inferred from the assigned value.

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main() {
8     var count int64
9     sum := count + 999
10    fmt.Println(sum)
11 }
12
```


Data Types in Go

- Integers: Explain the different integer types (e.g., int, int8, int16, int32, int64, uint, uint8, uint16, uint32, uint64) and their respective ranges.
- Floats: Discuss the float32 and float64 data types for representing floating-point numbers.
- Booleans: Explain the boolean data type, which can hold either true or false values.
- Strings: Describe strings as a sequence of characters enclosed in double quotes.

Basic Operations with Variables

- Arithmetic operations: Show examples of addition, subtraction, multiplication, division, and modulus operations on numerical variables.
- Concatenation: Illustrate how to concatenate strings using the + operator.
- Comparison operators: Introduce comparison operators (e.g., ==, !=, <, >, <=, >=) and demonstrate their usage in conditional statements.

Type Conversion - Basic Type

- Converting between numeric types: You can convert values between compatible numeric types like `int`, `float32`, `float64`, etc. For example: **`var x int = 5`** and **`var y float64 = float64(x)`**.
- Converting between integer types: You can convert values between different integer types, such as `int8`, `int16`, `int32`, `int64`, `uint8`, `uint16`, `uint32`, `uint64`, etc. For example: **`var a int16 = 10`** and **`var b int32 = int32(a)`**.

Type Conversion - String Conversion

Converting between string and numeric types: You can convert numeric values to a string representation using the `strconv` package's functions like `strconv.Itoa()` and `strconv.FormatFloat()`. Conversely, you can convert a string to a numeric type using functions like `strconv.Atoi()` and `strconv.ParseFloat()`.

Type Conversion - Type Conversion

Converting interface types: When working with interfaces, you may need to extract a specific underlying type from an interface. Type assertions allow you to do this. For example: **value := interfaceVar.(int).**

Type Conversion

```
1 package main
2
3 import (
4     "fmt"
5     "strconv"
6 )
7
8 func main() {
9     // Basic type conversion
10    var x int = 5
11    var y float64 = float64(x)
12    fmt.Printf("x: %d, y: %.2f\n", x, y)
13
14    // String conversion
15    var numString = "10"
16    num, err := strconv.Atoi(numString)
17    if err != nil {
18        fmt.Println("Error converting string to integer:", err)
19    } else {
20        fmt.Printf("num: %d\n", num)
21    }
22
23    // Type assertion
24    var val interface{} = 42
25    result, ok := val.(int)
26    if ok {
27        fmt.Println("Type assertion successful:", result)
28    } else {
29        fmt.Println("Type assertion failed")
30    }
31 }
32
```

Constants

- Constants are like variables but their values cannot be changed once defined.
- Declaring constants: `const pi = 3.14`
- Constants must be assigned a value at the time of declaration.

Zero Values

- When a variable is declared without an explicit value, it is assigned a zero value depending on its type.
- Example: `var count int` will assign 0 to count, false to bool, and an empty string to string.

Type	Zero Value
string	<code>""</code> // empty string
int, int32, int64	0
float32, float64	0.0
bool	false
slice eg, <code>[]int</code>	<code>[]</code>
pointer eg, <code>*int</code>	<code><nil></code>
map eg. <code>map[int]string</code>	<code>map[]</code>

If Statements

```
1 if condition {  
2     // code to execute if the condition is true  
3 else if condition2 {  
4     // code to execute if the condition 2 is true  
5 } else {  
6     // code to execute if the condition is false  
7 }  
8
```

```
1 age := 18  
2 if age >= 18 {  
3     fmt.Println("You are an adult.")  
4 } else {  
5     fmt.Println("You are a minor.")  
6 }  
7
```

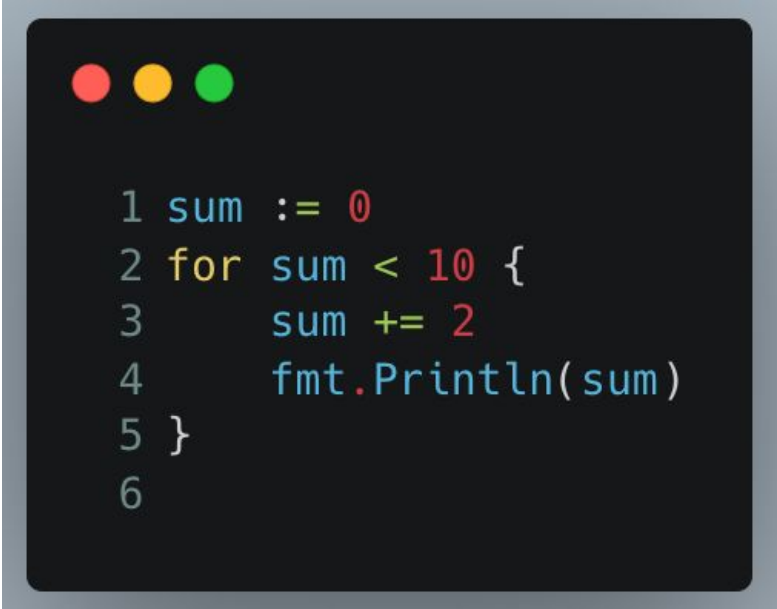
Switch Statements

A switch statement allows you to select one of several code blocks to execute based on the value of an expression.

```
1 day := "Monday"
2 switch day {
3 case "Monday", "Tuesday", "Wednesday", "Thursday", "Friday":
4     fmt.Println("It's a weekday.")
5 case "Saturday", "Sunday":
6     fmt.Println("It's a weekend.")
7 default:
8     fmt.Println("Invalid day.")
9 }
10
```

For Loops

A for loop is used to repeat a block of code a specified number of times or until a condition is met.

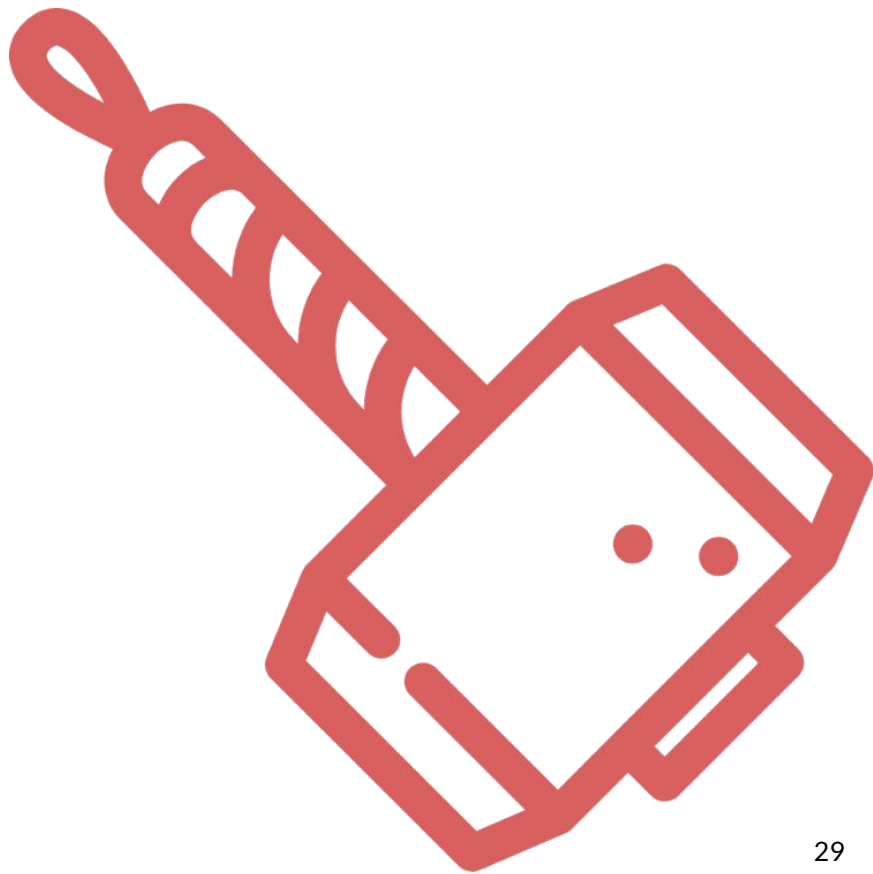


```
1 sum := 0
2 for sum < 10 {
3     sum += 2
4     fmt.Println(sum)
5 }
6
```

Demo

Demo - Zer0 to Hero

- Set up the basic Go project structure.
- Implement the server's initialization and basic routing.
- Create a simple "Hello, World!" endpoint to ensure the server is running correctly.



Assignment

Assignment for Day 1

Reordering Names Based on Country Code

Goal: Create a program that reorders names based on the specified country's format.

Inputs: First name, last name, middle name (optional), and country code.

Outputs: Reordered name based on the country's format.

A terminal window with a dark background and light text. The title bar shows three colored dots (red, yellow, green) and the word "bash". The terminal contains four lines of text: a command to run a Go program with "John Smith VN" as input, the output "Smith John", a second command with "Emily Rose Watson US" as input, and the output "Emily Rose Watson".

```
bash
1 $ go run reorder_names.go John Smith VN
2 Output: Smith John
3 $ go run reorder_names.go Emily Rose Watson US
4 Output: Emily Rose Watson
```

Assignment for Day 1

Parse command line arguments to extract the first name, last name, middle name (if provided), and the country code.

Determine the name format based on the country code.

Reorder the name according to the determined format.

Output the reordered name.

Assignment for Day 1 - Hint

Use the `os` package to access command line arguments `os.Args`.

Handle cases where the middle name is not provided.

Consider using a switch statement to determine the name format based on the country code.

Reference

Resources & Reference links

- <https://go.dev/tour/basics/1>

Thank You



Q&A

