UNIT-1

Getting started, Machine Setup - Text Editors, The Terminal, Environment, Go, Your First Program, How to Read a Go Program, Numbers - Integers, Floating-Point Numbers, Example - Strings, Booleans.

- Go, also known as Golang, is a statically-typed, compiled programming language developed by Google.
- It was designed to be efficient, simple, and scalable, making it suitable for building various types of applications, including web services, command-line tools, and networked servers.
- Go incorporates features from other programming languages while also introducing its own unique concepts.

Key aspects of the Go programming language:

Simplicity: Go aims to keep the language simple and minimalistic, with a small number of keywords and a straightforward syntax. It emphasizes readability and ease of maintenance.

Concurrency: Go has built-in support for concurrency and parallelism. Goroutines are lightweight threads that allow concurrent execution of functions, enabling efficient utilization of multiple CPU cores. The language provides channels for communication and synchronization between goroutines.

Strong typing: Go is statically typed, meaning that variable types are checked at compile time. This helps catch errors early in the development process and improves the overall reliability of the code.

Garbage collection: Go includes automatic memory management through garbage collection. Developers do not need to explicitly allocate or deallocate memory, reducing the risk of memory leaks and making memory management easier.

Standard library: Go comes with a rich standard library that provides a wide range of functionality, including networking, file I/O, cryptography, JSON encoding/decoding, and much more. The standard library is well-documented and is an essential resource for Go developers.

Compilation and execution: Go code is compiled into machine code, resulting in highly efficient and performant executables. Go binaries can be easily distributed and run on different platforms without requiring the target system to have Go installed.

Tooling: Go provides a set of command-line tools, including the go command, which facilitates tasks such as code formatting, building, testing, and dependency management. The go command also supports module-based development, enabling better dependency management.

How Go lang programming is different from c, c++, python, java

Go programming language differs from C, C++, Python, and Java in several ways. Here are some of the key differences:

Syntax: Go has a simpler and more concise syntax compared to C and C++. It eliminates certain features like header files, preprocessor directives, and complex pointer arithmetic, making the code easier to read and maintain.

Concurrency: Go has built-in support for concurrency with goroutines and channels. Goroutines allow lightweight concurrent execution, while channels facilitate communication and synchronization between goroutines. In contrast, C, C++, Python, and Java typically require explicit threading or use higher-level abstractions for concurrency.

Memory Management: Go has a garbage collector that automatically manages memory allocation and deallocation. Developers don't need to manually allocate or free memory as they would in C or C++. Python and Java also have garbage collectors, but C and C++ require manual memory management.

Type System: Go has a static type system like C, C++, and Java, where types are checked at compile time. However, Go's type system is simpler and more limited compared to C++ and Java. Python, on the other hand, is dynamically typed, meaning types are checked at runtime.

Standard Library: Go comes with a comprehensive standard library that provides a wide range of functionality out of the box. The standard libraries of C, C++, Python, and Java are also extensive, but they differ in terms of specific features and APIs available.

Compilation and Execution: Go is a compiled language, like C and C++, but it has a faster compilation process compared to C++. Go binaries are statically linked, resulting in standalone executables without external dependencies. Python and Java, on the other hand, are interpreted or JIT-compiled languages.

Error Handling: Go uses explicit error handling with return values, similar to C. This approach encourages developers to handle errors explicitly instead of relying on exceptions, which are commonly used in C++, Python, and Java.

Object-Oriented Programming: Go supports object-oriented programming (OOP) concepts, but it does not have traditional classes and inheritance like C++, Java, and Python. Instead, Go has struct types with associated methods, interfaces, and composition to achieve code reuse and polymorphism.

Go lang programming - Getting Started

Install Go: First, you need to download and install Go on your system. Visit the official Go website (https://golang.org/dl/) and download the installer appropriate for your operating system. Run the installer and follow the instructions to install Go.

Set up your workspace: Go requires a specific directory structure called a workspace. Create a directory anywhere you like on your system, and set it as your Go workspace. You can do this by setting the *GOPATH* environment variable to the path of your workspace directory.

For example, if you create a directory called "go" in your user's home directory, you would set the *GOPATH* variable as follows:

```
Export GOPATH=$HOME/go
```

It's also recommended to add the Go binary directory to your system's PATH variable. For Unix-like systems, add the following line to your shell profile file (e.g., ~/.bashrc or ~/.bash_profile):

```
export PATH=$PATH:/usr/local/go/bin:$GOPATH/bin
```

Create your first Go program: Now you can start writing your first Go program. Create a new file with a .go extension, for example, hello.go. Open the file in a text editor and enter the following code:

```
package main

import "fmt"

func main() {

fmt.Println("Hello, World!")

}
```

Build and run the program: Open a terminal or command prompt, navigate to the directory containing your hello.go file, and run the following command to build the Go program:

```
go build
```

This will create an executable file in the current directory.

To run the program, execute the generated executable:

```
./hello
```

Go lang Machine setup - Text Editors, The Terminal, Environment

Setting up your machine for Go programming involves choosing a text editor or integrated development environment (IDE), configuring the terminal, and understanding the Go environment.

Text Editors and IDEs:

Visual Studio Code (VS Code): VS Code is a popular and lightweight code editor with excellent support for Go. Install the Go extension for VS Code, which provides features like code completion, formatting, and debugging for Go programs.

GoLand: GoLand is a dedicated IDE by JetBrains specifically designed for Go development. It offers a comprehensive set of features, including advanced code analysis, refactoring tools, and integrated version control support.

Apart from these options, other editors such as Sublime Text, Atom, and Vim also have Go plugins or packages available. Choose the one that suits your preferences and workflow.

The Terminal: Go programming relies heavily on the command-line interface. You can use the built-in terminal or a separate terminal emulator application.

Built-in Terminal: Most text editors and IDEs, including VS Code and GoLand, provide an integrated terminal within the application. You can open a terminal window directly within the editor and execute Go commands.

Separate Terminal Emulator: If you prefer a separate terminal emulator, popular options include Terminal (macOS), Git Bash (Windows), iTerm2 (macOS), and GNOME Terminal (Linux). These emulators provide a command-line interface where you can execute Go commands.

Go Environment: The Go environment consists of several important environment variables and directory structures.

GOPATH: The GOPATH environment variable specifies the location of your Go workspace. It should point to the directory where you store your Go projects. Make sure to set this variable correctly as explained in the previous response.

GOROOT: The GOROOT environment variable specifies the location where Go is installed on your machine. This variable is automatically set when you install Go using the official installer, so you typically don't need to configure it manually.

PATH: Add the Go binary directory to your system's PATH variable so that you can execute Go commands from any location in the terminal. For Unix-like systems, update your shell profile file (e.g., ~/.bashrc or ~/.bash_profile) with the following line:

```
export PATH=$PATH:/usr/local/go/bin:$GOPATH/bin
```

Understanding and correctly configuring these environment variables is essential for a smooth Go development experience.

By selecting an appropriate text editor or IDE, setting up a terminal, and configuring the Go environment, you'll be well-prepared to start writing and running Go programs on your machine.

Go, First program

```
package main


import "fmt"


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

To run this program:

Save the code in a file with a .go extension, such as hello.go.

Open a terminal or command prompt, navigate to the directory where you saved the file.

Run the following command to build the Go program:

```
go build hello.go
```

This will create an executable file named hello in the same directory.

```
./hello
```

You should see the output Hello, World! printed to the console.

This program demonstrates the basic structure of a Go program. The package main statement at the beginning indicates that this is the main package, which is required for executable programs. The import "fmt" statement imports the "fmt" package, which provides functionality for formatted I/O. The main() function is the entry point of the program, and fmt.Println("Hello, World!") prints the text "Hello, World!" to the console.

How to read a Go Program

To read and understand a Go program, you can follow these steps:

Analyze the package declaration: Every Go program starts with a package declaration. It specifies the package name that the program belongs to. For example, package main indicates that this program is part of the main package, which is the entry point for executable programs.

Review the imports: After the package declaration, you'll find import statements that specify external packages used in the program. These packages provide additional functionality that the program relies on. For example, import "fmt" imports the "fmt" package, which is used for formatted I/O operations.

Identify the main function: The main() function is the entry point of the program. It is where the program execution begins. Look for the func main() declaration. The code inside this function will be executed when the program runs.

Study the program logic: Read and analyze the code within the main() function and any other functions called from it. Understand the purpose and flow of the program. Identify control structures like conditionals (if, else, switch) and loops (for) that control the program's behavior.

Inspect function calls: Look for function calls within the program. Understand what each function does and how it contributes to the program's functionality.

Examine variable declarations and assignments: Pay attention to variable declarations (var) and assignments (=). Understand the purpose and usage of variables within the program. Consider the types of variables and how they are initialized or modified.

Read comments: Go programs often include comments that provide explanations or documentation. Read and understand the comments to gain insights into the code's purpose and behavior.

Refer to the Go documentation: Whenever you encounter unfamiliar syntax, functions, or types, refer to the official Go documentation (https://golang.org/doc/) for detailed explanations and examples. The documentation is a valuable resource for understanding the Go programming language and its standard library.

By following these steps and gradually analyzing different aspects of the program, you can develop a better understanding of how a Go program works. Don't hesitate to experiment with the code, make modifications, and observe the resulting behavior to deepen your understanding of Go programming concepts.

Numbers - Integers in Go lang programming

In Go programming, integers are a fundamental data type used to represent whole numbers without fractional parts. Go provides various integer types with different sizes and ranges to accommodate different needs. Here are the commonly used integer types in Go:

Signed Integers:

int8: Signed 8-bit integers ranging from -128 to 127.

int16: Signed 16-bit integers ranging from -32,768 to 32,767.

int32 or rune: Signed 32-bit integers ranging from -2,147,483,648 to 2,147,483,647. The rune type is an alias for int32 and is commonly used to represent Unicode code points.

int64: Signed 64-bit integers ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

Unsigned Integers:

uint8 or byte: Unsigned 8-bit integers ranging from 0 to 255. The byte type is an alias for uint8 and is commonly used to represent ASCII characters.

uint16: Unsigned 16-bit integers ranging from 0 to 65,535.

uint32: Unsigned 32-bit integers ranging from 0 to 4,294,967,295.

uint64: Unsigned 64-bit integers ranging from 0 to 18,446,744,073,709,551,615.

Platform-Dependent Integers:

int: The int type is a signed integer that is platform-dependent, meaning its size and range depend on the underlying system architecture. It typically matches the size of the system's word size (e.g., 32-bit or 64-bit).

Unsigned Integer with Platform-Dependent Size:

uint: The uint type is an unsigned integer that is platform-dependent, similar to the int type. Its size matches the size of the system's word size.

When declaring integer variables, you can use the var keyword followed by the variable name and its type. Here's an example:

```
var num1 int32 = 42

var num2 uint8 = 255
```

In this example, num1 is a signed 32-bit integer with a value of 42, and num2 is an unsigned 8-bit integer with a value of 255.

Go provides a rich set of arithmetic and bitwise operations to work with integers, allowing you to perform calculations, comparisons, and bitwise manipulations. You can also convert between different integer types using explicit type casting.

Remember to consider the size and range of integer types when choosing the appropriate type for your program to ensure data integrity and efficient memory usage.

Floating - Point numbers in Go lang programming

In Go programming, floating-point numbers are used to represent real numbers with fractional parts. Go provides two floating-point types: float32 and float64, representing 32-bit and 64-bit floating-point numbers, respectively.

Floating-point number types in Go:

float32: This type represents single-precision floating-point numbers, also known as "floats." It stores numbers with a precision of about 7 decimal digits. The range of float32 values is approximately ±1.18e-38 to ±3.4e38.

float64: This type represents double-precision floating-point numbers, often referred to as "doubles." It provides higher precision compared to float32 and can store numbers with about 15 decimal digits of precision. The range of float64 values is approximately ±2.23e-308 to ±1.8e308.

When declaring floating-point variables, you can use the var keyword followed by the variable name and its type.

```
Var num1 float32=3.14

Var num2 float64=1.23456789
```

In this example, num1 is a float32 variable initialized with the value 3.14, and num2 is a float64 variable initialized with the value 1.23456789.

- Go supports various arithmetic and mathematical operations on floating-point numbers, such as addition, subtraction, multiplication, division, and more.
- It's important to note that floating-point calculations may not always produce exact results due to the inherent limitations of representing real numbers in a binary format.
- Rounding errors and precision limitations can occur, so it's important to handle floating-point calculations with care, especially when comparing values for equality.

To format floating-point numbers for output, you can use the fmt.Printf function or other formatting options provided by the fmt package.

For example:

```
num:=3.14159

fmt.Printf("%.2f\n", num)//Output:3.14
```

This example uses the format verb %f to print the floating-point number with two decimal places.

When working with floating-point numbers, it's important to understand their limitations and consider the precision required for your specific use case. In situations where precise decimal calculations are needed, such as financial applications, it's recommended to use decimal arithmetic libraries specifically designed for such purposes.

Example - Strings, Booleans in Go lang programming

Strings:

```
package main

import "fmt"

func main() {

   // Declare a string variable

   var message string = "Hello, World!"
```

```
    // Print the string
    fmt.Println(message)
   // Concatenate strings
    name := "Alice"
    greeting := "Hello, " + name + "!"
    fmt.Println(greeting)
  // Get the length of a string
    fmt.Println("Length of 'Hello':", len("Hello"))


    // Access individual characters of a string
    fmt.Println("First character of 'Golang':", "Golang"[0])
}
```

In this example, we declare a string variable message and assign it the value "Hello, World!". We then use fmt.Println to print the string to the console.

We also demonstrate string concatenation by combining the name variable with the greeting message. The len function is used to get the length of the string "Hello". Lastly, we access individual characters of the string "Golang" using indexing.

Booleans:

```
package main
import "fmt"
func main() {
    // Declare boolean variables
    var isTrue bool = true
    var isFalse bool = false


    // Print boolean values
    fmt.Println("isTrue:", isTrue)
    fmt.Println("isFalse:", isFalse)
    // Perform boolean operations
    result := isTrue && isFalse
    fmt.Println("isTrue && isFalse:", result)


    result = isTrue || isFalse
    fmt.Println("isTrue || isFalse:", result)


    result = !isTrue
    fmt.Println("!isTrue:", result)
}
```

In this example, we declare boolean variables isTrue and isFalse and assign them boolean values. We then use fmt.Println to print the boolean values to the console.

We also demonstrate boolean operations such as logical AND (&&), logical OR (||), and logical NOT (!). The results of these operations are assigned to the result variable and printed to the console.

These examples showcase basic usage of strings and booleans in Go programming. Strings are used to represent text, and various operations can be performed on them. Booleans, on the other hand, represent true or false values and are commonly used in conditions and logical operations.