Guide to Go

Table of Contents

1. Abstract

The purpose of this technical paper is to provide a concise and comprehensive introduction to the Go programming language, with a focus on practical application. One will be guided through the process of installing and configuring a Go development environment, writing and executing their first program as well as understanding the fundamental concepts of the language. Key topics include basic syntax and control structures, package management, error handling, and the principles of Go's concurrency model. Each section is supported by runnable examples available in the accompanying Git repository, allowing readers to apply concepts immediately and build a solid foundation for Go.

2. Why Go?

Go originated in mid to late 2007 at the Google headquarters. The so called founding fathers were Robert Griesemer, Rob Pike and Ken Thompson. They were stumped by an engineering challenge. Rather tackling the issue itself, they decided to innovate and create Go. This was due to the fact that there was no other programming language that they could use which would fit their criteria. After the project grew and more employees from Google joined the team, the first stable version was released in March of 2012. Today Go can be seen being used by Google, Microsoft, PayPal, American Express, Netflix, Uber, X, Twitch and many more.
Go offers simplicity alongside a simple syntax. This included a fast compiler, concurrency and tooling. Go is designed for modern Hardware whereas programming languages in 2007, such as C, C++, JAVA and Python were not.

3. Introduction to Go

There are many ways to download Go. Let’s start with the easiest and the simplest. Go and install it from their download page. For Apple users, please note that the download file needs to align with the processor you have, whether its ARM64 (M-Chip) or an intel chip. This is an alternative if the direct install from the website does not work.

For MacOS Intel based

```
curl -LO https://go.dev/dl/go1.25.0.darwin-amd64.pkg && sudo installer -pkg
go1.25.0.darwin-amd64.pkg -target /
```

Auto check for base architecture

```
ARCH=$(uname -m) && if [ "$ARCH" = "x86_64" ]; then PKG=go1.25.0.darwin-amd64.pkg;
else PKG=go1.25.0.darwin-arm64.pkg; fi && curl -LO https://go.dev/dl/$PKG && sudo
installer -pkg $PKG -target /
```

For Windows in Git Bash

```
curl -LO https://go.dev/dl/go1.25.0.windows-amd64.msi && cmd //c start ""
go1.25.0.windows-amd64.msi
```

Once go has been installed make sure you have the correct version (1.25.0) installed by checking it with:

```
go version
```

Now we need a suitable IDE for the actual coding part of this paper. I will be using GoLand by JetBrains, however there are many IDEs that support Go, but a simpler IDE is also Visual Studio Code. One needs to install the plugins for the respected IDE for Go to work properly.

4. Go Basics

A little Tip: One needs to run the "Go Tutorial" file so that everything runs smoothly because Go runs file one at a time, therefore we can not run "Go.main" and expect the "basics()" function to be recognized / defined.

Go is based off of C therefore it will have similar types and variables, however they are quite the same throughout all programming languages. Before we can start the actual development with Go, there are a few things we have to look at namely:

1. Variables, constants, and types
    a. Constants

   i. Constants are declared at package level, therefore even before we start with our main functions.

```
package main

import (
  "fmt"
)

const (
    PI              = 3.14159
    Min             = 55
    AppName         = "Basics of Go"
    IsPublic        = false
)
```

1. Basics
   i. These include Strings, integers, floats etc. These are declared in the funcions themselves
   ii. There are many ways and tricks to declare these variables, this paper covers the most common and practical methods

```
func main() {


    // Method 1: full
    var name string = "John"
    var age int = 30
    var salary float64 = 50000.50
    var isActive bool = true

    // Method 2: inferred
    var city = "New York"
    var number = 10.0


    // Method 3: short declaration (most common)
    country := "USA"
    zipCode := 10001

    // Method 4: Multiple variable declaration
    var (
        firstName string = "Jane"
        lastName  string = "Doe"
        userID    int    = 12345
    )

    // Method 5: Multiple assignment
    x, y := 10, 20
    a, b, c := 1, 2.5, "hello"
```

a. Arrays and Slices

   i. Arrays in Go have a fixed length and contain elements of the same type. Once declared, their size cannot be changed. Slices, on the other hand, are built on top of arrays but are far more flexible as they can grow and shrink as needed.

```
func main() {

    //array
    var numbers [5]int   // 0 by default
    numbers[0] = 1
    numbers[1] = 2
    fmt.Println(numbers)               // Output: [1 2 0 0 0]

    // Array with initialization
    primes := [5]int{2, 3, 5, 7, 11}

    // Slices
    var fruits []string
    fruits = append(fruits, "Apple", "Banana")
    fmt.Println(fruits)                 // Output: [Apple Banana]

    // Slice from array
    slicePrimes := primes[1:4]
    fmt.Println(slicePrimes)            // Output: [3 5 7]
}
```

1. Control structures (if, for, switch)

   i. Go offers simple and readable control structures. There is no `while` keyword, the `for` loop covers all looping needs.

```
func main() {
    // If-else
    age := 18
    if age >= 18 {
        fmt.Println("adult")
    } else {
        fmt.Println("under age")
    }

    // For loop
    for i := 0; i < 5; i++ {
        fmt.Println(i)
    }

    // For as while
    sum := 1
    for sum < 100 {
        sum += sum
    }

    // Switch
    day := "Monday"
    switch day {
    case "Monday":
        fmt.Println("Start of the week")
    case "Friday":
        fmt.Println("Almost weekend")
    default:
        fmt.Println("Midweek day")
    }
}
```

1. Functions (including multiple return values)

   i. Functions in Go are declared with the `func` keyword and can return multiple values, which is a common pattern in Go programs.

```
func add(a int, b int) int {
    return a + b
}

func divide(a, b int) (int, int) {
    return a / b, a % b
}

func main() {
    fmt.Println(add(3, 4))             // Output: 7

    quotient, remainder := divide(10, 3)
    fmt.Println(quotient, remainder)   // Output: 3 1
}
```

1. Scope and naming conventions

   i. Variables in Go have scope based on where they are declared. Package-level variables are accessible throughout the package, while function-level variables are local to that function. Exported identifiers (accessible from other packages) start with an uppercase letter.

```
package main

import "fmt"

// Package-level variable (exported)
var Version = "1.0"
```

```
// Package-level variable (unexported)
var internalID = 123

func main() {
    localVar := "I am local"
    fmt.Println(localVar)
    fmt.Println(Version)
}
```

5. Your First Go Program

```go
package main

import "fmt"

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

- How to compile and run
- Explanation of basic syntax

6. Structs and Interfaces

- Defining and using `struct`
- Methods and receivers
- Interfaces and polymorphism

7. Error Handling in Go

- Idiomatic error checking
- Using the `errors` package
- Custom error types (optional)

8. Go Concurrency

- Goroutines: launching lightweight threads
- Channels: communication between goroutines
- `select` keyword
- Sample: worker pool example

9. Project Structure and Go Modules

- Using `go mod` properly
- Recommended folder layout
- Dependency management

10. Testing in Go

- Using the `testing` package
- Writing and running unit tests
- Example with table-driven tests

11. Hashing and Websockets

- Community links (Go.dev, Go by Example, Effective Go)
- Tools (`go fmt`, `golint`, `go vet`)
- Where to go next (web frameworks, microservices, etc.)

12. Appendix

- Link to GitHub Repository
- File structure
- System requirements

13. Sources

1. Go-How It All Began *by Medium*
2. Wikipedia
3. Go Web Examples
4. Go.dev