Go Variables Tutorial

Variables are fundamental building blocks in Go programming. They store data that can be used and modified throughout your program. This tutorial covers everything you need to know about variables in Go.

Table of Contents

Variable Declaration

In Go, variables must be declared before use. The basic syntax is:

```
var variableName dataType
```

Examples:

```
package main

import "fmt"

func main() {
    var name string
    var age int
    var height float64
    var isStudent bool

    fmt.Println(name, age, height, isStudent)
    // Output: "" 0 0 false (zero values)
}
```

Data Types

Go has several built-in data types:

Numeric Types

```
// Integers
```

```
var a int = 42           // Platform-dependent size (32 or 64 bit)
var b int8 = 127         // 8-bit signed integer (-128 to 127)
var c int16 = 32767      // 16-bit signed integer
var d int32 = 2147483647 // 32-bit signed integer
var e int64 = 9223372036854775807 // 64-bit signed integer

// Unsigned integers
var f uint = 42          // Platform-dependent size
var g uint8 = 255        // 8-bit unsigned (0 to 255)
var h uint16 = 65535     // 16-bit unsigned
var i uint32 = 4294967295 // 32-bit unsigned
var j uint64 = 18446744073709551615 // 64-bit unsigned

// Floating-point numbers
var k float32 = 3.14     // 32-bit floating point
var l float64 = 3.141592653589793 // 64-bit floating point

// Complex numbers
var m complex64 = 1 + 2i   // Complex number with float32 parts
var n complex128 = 1 + 2i  // Complex number with float64 parts
```

String Type

```
var message string = "Hello, Go!"
var multiline string = `This is a
multi-line
string`
```

Boolean Type

```
var isReady bool = true
var isComplete bool = false
```

Pointer Types

```
var ptr *int        // Pointer to an integer
var x int = 42
ptr = &x           // ptr now points to x
fmt.Println(*ptr)  // Dereference ptr to get the value (42)
```

Zero Values

In Go, variables are automatically initialized to their zero values if not explicitly initialized:

```
var i int       // 0
var f float64   // 0.0
var b bool      // false
var s string    // ""
var p *int      // nil (for pointers)
```

Variable Initialization

You can declare and initialize variables in several ways:

Method 1: Declare then assign

```
var name string
name = "Alice"
```

Method 2: Declare and initialize

```
var name string = "Alice"
var age int = 25
```

Method 3: Type inference

```
var name = "Alice"  // Go infers string type
var age = 25        // Go infers int type
```

Short Variable Declaration

The `:=` operator provides a concise way to declare and initialize variables:

```
func main() {
    name := "Bob"        // string
    age := 30           // int
    height := 5.9       // float64
    isMarried := true   // bool

    fmt.Println(name, age, height, isMarried)
}
```

Important Notes:

- Short declaration can only be used inside functions
- At least one variable on the left side must be new
- Cannot be used to redeclare variables at package level

Multiple Variable Declaration

Group Declaration

```
var (
    name     string = "Charlie"
    age      int    = 35
    salary   float64 = 50000.0
    isActive bool   = true
)
```

Multiple Assignment

```
// Same type
var a, b, c int = 1, 2, 3

// Different types
var name, age, salary = "Diana", 28, 45000.0
```

```
// Using short declaration
x, y, z := 10, 20, 30
```

Multiple Return Values

```
func getPersonInfo() (string, int) {
    return "Eve", 32
}

func main() {
    name, age := getPersonInfo()
    fmt.Println(name, age)

    // Ignore a return value with blank identifier
    name, _ := getPersonInfo()
    fmt.Println(name)
}
```

Constants

Constants are immutable values that cannot be changed after declaration:

```
const Pi = 3.14159
const MaxUsers = 1000
const AppName = "MyApp"

// Group constants
const (
    StatusActive   = 1
    StatusInactive = 0
    StatusPending  = 2
)

// iota for auto-incrementing constants
const (
    Sunday = iota    // 0
    Monday           // 1
    Tuesday          // 2
    Wednesday        // 3
    Thursday         // 4
    Friday           // 5
    Saturday         // 6
)
```

Variable Scope

Package-level Variables

```
package main

import "fmt"

var globalVar = "I'm global"  // Accessible throughout the package
```

```
func main() {
    fmt.Println(globalVar)
    someFunction()
}

func someFunction() {
    fmt.Println(globalVar)  // Can access global variable
}
```

Function-level Variables

```
func main() {
    localVar := "I'm local"  // Only accessible within main function
    fmt.Println(localVar)

    if true {
        blockVar := "I'm in a block"  // Only accessible within this block
        fmt.Println(localVar)         // Can access function-level variable
        fmt.Println(blockVar)
    }

    // fmt.Println(blockVar)  // Error: blockVar not accessible here
}
```

Type Conversion

Go requires explicit type conversion between different types:

```
func main() {
    var i int = 42
    var f float64 = float64(i)  // Convert int to float64
    var u uint = uint(f)        // Convert float64 to uint

    // String conversions
    var s string = string(rune(65))  // "A" (ASCII 65)

    // For string to number conversions, use strconv package
    import "strconv"

    var numStr = "123"
    num, err := strconv.Atoi(numStr)  // String to int
    if err != nil {
        fmt.Println("Conversion error:", err)
    } else {
        fmt.Println("Number:", num)
    }

    var floatStr = "3.14"
    floatNum, err := strconv.ParseFloat(floatStr, 64)
    if err != nil {
        fmt.Println("Conversion error:", err)
    } else {
        fmt.Println("Float:", floatNum)
    }
}
```

Best Practices

1. Use Meaningful Variable Names

```
// Bad
var d int = 30
var u string = "john@example.com"

// Good
var daysInMonth int = 30
var userEmail string = "john@example.com"
```

2. Use Short Declaration When Possible

```
// Less preferred
var name string = "Alice"
var age int = 25

// Preferred
name := "Alice"
age := 25
```

3. Group Related Variables

```
var (
    serverHost = "localhost"
    serverPort = 8080
    serverTimeout = 30
)
```

4. Use Constants for Fixed Values

```
const (
    MaxRetries = 3
    DefaultTimeout = 30
    APIVersion = "v1"
)
```

5. Initialize Variables Close to Usage

```
func processUser() {
    // Initialize when needed
    user := getCurrentUser()
    if user == nil {
        return
    }

    // Use the variable immediately
    fmt.Println("Processing user:", user.Name)
}
```

6. Use Zero Values When Appropriate

```
// Instead of explicitly initializing to zero values
```

```
var count int = 0
var message string = ""

// Just declare and use zero values
var count int
var message string
```

Common Pitfalls

1. Unused Variables

Go compiler will give an error for unused variables:

```
func main() {
    name := "Alice"  // Error: name declared but not used
    age := 25
    fmt.Println(age)
}
```

Use the blank identifier _ if you need to ignore a value:

```
func main() {
    _, age := getPersonInfo()  // Ignore name, use only age
    fmt.Println(age)
}
```

2. Variable Shadowing

```
var x = 10

func main() {
    fmt.Println(x)  // Prints 10

    if true {
        x := 20     // This creates a new variable, shadowing the global x
        fmt.Println(x)  // Prints 20
    }

    fmt.Println(x)  // Prints 10 (global x unchanged)
}
```

Summary

Variables in Go are strongly typed and must be declared before use. Key points to remember:

- Use `var` for explicit declaration or `:=` for short declaration
- Go provides automatic zero values for uninitialized variables
- Type conversion must be explicit
- Variables have different scopes (package, function, block)
- Use meaningful names and follow Go conventions
- Constants are immutable and useful for fixed values

Understanding variables is crucial for Go programming. Practice with different types, scopes, and declaration methods to become proficient in Go development.