

A Whirlwind Tour of Go

Just the Cool Parts

Steve Willoughby

10-Apr-2024

v0.0.1



The Point

- “What *is* Go?”
- “What is it actually good for?”
- “Why should I care?”

30 Seconds of History

- Designed by Rob Pike, Ken Thompson, and Robert Griesemer.
- Includes direct experience with C from day 1 to now.

30 Seconds of History

- Designed by Rob Pike, Ken Thompson, and Robert Griesemer.
- Includes direct experience with C from day 1 to now.
- “If we were to design C today, knowing what we know now, on today’s technology...”

30 Seconds of History

- Designed by Rob Pike, Ken Thompson, and Robert Griesemer.
- Includes direct experience with C from day 1 to now.
- “If we were to design C today, knowing what we know now, on today’s technology...”
 - ∴ Go’s syntax is very much like C’s
 - ... but cleaned up and streamlined a bit.

30 Seconds of History

- Designed by Rob Pike, Ken Thompson, and Robert Griesemer.
- Includes direct experience with C from day 1 to now.
- “If we were to design C today, knowing what we know now, on today’s technology...”
 - ∴ Go’s syntax is very much like C’s
 - ... but cleaned up and streamlined a bit.
- Dreamed up while waiting on a 45-minute C⁺⁺ compile

30 Seconds of History

- Designed by Rob Pike, Ken Thompson, and Robert Griesemer.
- Includes direct experience with C from day 1 to now.
- “If we were to design C today, knowing what we know now, on today’s technology...”
 - ∴ Go’s syntax is very much like C’s
 - ... but cleaned up and streamlined a bit.
- Dreamed up while waiting on a 45-minute C⁺⁺ compile
 - Fast compilation
 - Native binary compiler with low overhead
 - Strong static typing
 - Extraordinarily spartan

Go Syntax

- Type declarations *follow* identifier names

```
var x int
```

```
var UserName string
```

```
func AddNumbers(x, y int) int { ... }
```

```
func DivideNumbers(x, y int) (int, error) { ... }
```

```
type Shape struct {
```

```
    X      int
```

```
    Y      int
```

```
    Color  ColorCode
```

```
}
```


Program Structure

- Basic unit is a *package* (namespace boundary).

Program Structure

- Basic unit is a *package* (namespace boundary).
- Multiple source files in a package, in the same directory tree.

Program Structure

- Basic unit is a *package* (namespace boundary).
- Multiple source files in a package, in the same directory tree.
- Every program must have a `main` package.
- The `main` package has a `main` function.

Program Structure

- Basic unit is a *package* (namespace boundary).
- Multiple source files in a package, in the same directory tree.
- Every program must have a `main` package.
- The `main` package has a `main` function.
- Import packages into the program using the `import` statement.
- Always prefix identifiers from imported packages with their package name.

Program Structure

- Basic unit is a *package* (namespace boundary).
- Multiple source files in a package, in the same directory tree.
- Every program must have a `main` package.
- The `main` package has a `main` function.
- Import packages into the program using the `import` statement.
- Always prefix identifiers from imported packages with their package name.
- Identifiers can be *public* or *private* w/r/t package boundaries.

Program Structure

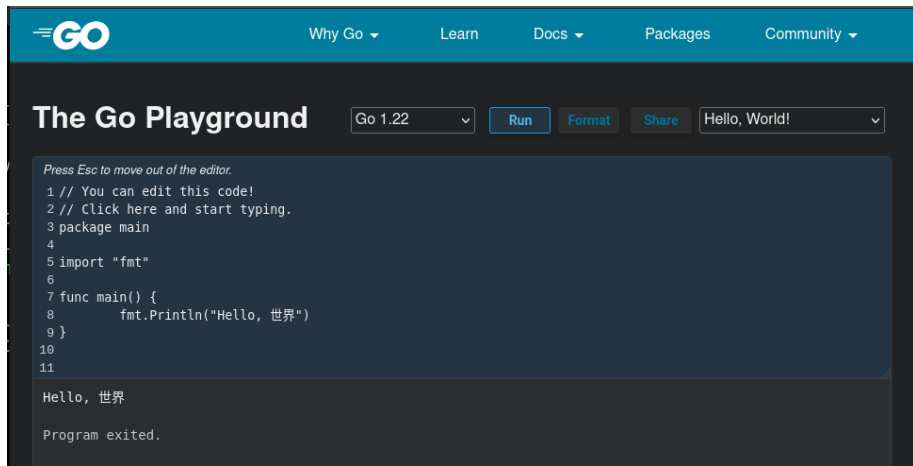
- Basic unit is a *package* (namespace boundary).
- Multiple source files in a package, in the same directory tree.
- Every program must have a `main` package.
- The `main` package has a `main` function.
- Import packages into the program using the `import` statement.
- Always prefix identifiers from imported packages with their package name.
- Identifiers can be *public* or *private* w/r/t package boundaries.
 - Identifier names starting with an uppercase letter are public.
 - All others are private.

Hello, World

```
/* Standard-issue "Hello, World" program in Go */  
  
package main  
  
import "fmt"  
  
func main() {  
    fmt.Println("Hello, 世界")  
}
```

The Playground

- Interactive playground to immediately try something in Go.
- <https://go.dev/play/>



The screenshot shows the Go Playground web interface. At the top is a teal navigation bar with the Go logo and links for 'Why Go', 'Learn', 'Docs', 'Packages', and 'Community'. Below this is a dark grey header area with the title 'The Go Playground', a version selector set to 'Go 1.22', and buttons for 'Run', 'Format', and 'Share'. To the right of these buttons is a dropdown menu showing 'Hello, World!'. The main area is a code editor with a dark background. It contains the following Go code:

```
1 // You can edit this code!
2 // Click here and start typing.
3 package main
4
5 import "fmt"
6
7 func main() {
8     fmt.Println("Hello, 世界")
9 }
10
11
```

Below the code editor, the output of the program is displayed: 'Hello, 世界' followed by 'Program exited.' on a new line.

Importing Third-Party Packages

- Standard library package names are simple names:

```
import "fmt"  
import "encoding/json"  
import "flag"  
import "math"
```

Importing Third-Party Packages

- Standard library package names are simple names:

```
import "fmt"  
import "encoding/json"  
import "flag"  
import "math"
```

- Getting packages from public repositories:

```
import "github.com/MadScienceZone/go-gma/v5/dice"
```

Automatic API Documentation

• <https://pkg.go.dev/repository-url>

The screenshot shows the Go Package Documentation page for the 'dice' package. The page has a dark theme with a teal header. The header includes the Go logo, a search bar, and navigation links: 'Why Go', 'Learn', 'Docs', 'Packages' (active), and 'Community'. Below the header, the breadcrumb 'Discover Packages > github.com/MadScienceZone/go-gma/v5 > dice' is shown. The package name 'dice' is highlighted with a 'package' tag. Metadata includes 'Version: v5.17.0' (marked as 'Latest'), 'Published: Feb 28, 2024', 'License: BSD-3-Clause', 'Imports: 16', and 'Imported by: 0'. A section with 'Details', 'Repository', and 'Links' provides additional information. The 'Documentation' section is expanded, showing a sidebar with 'Overview', 'Index', 'Constants', 'Variables', 'Functions', 'Types', and 'Source Files'. The 'Overview' content describes the package's purpose and provides example code for using the 'Roll' and 'RollOnce' functions, as well as the 'NewDieRoller' function.

GO Search packages or symbols Why Go Learn Docs Packages Community

Discover Packages > github.com/MadScienceZone/go-gma/v5 > dice

dice package

Version: v5.17.0 **Latest** | Published: Feb 28, 2024 | License: BSD-3-Clause | Imports: 16 | Imported by: 0

Details Valid go.mod file Redistributable license Tagged version Stable version Learn more about best practices

Repository github.com/MadScienceZone/go-gma

Links Open Source Insights

Jump to ...

Documentation

Overview
Index
Constants
Variables
Functions
Types
Source Files

<> Documentation

Overview

Package dice provides a general facility for generating random numbers in fantasy role-playing games.

The preferred usage model is to use the higher-level abstraction provided by DieRoller, which rolls dice as described by strings. For example:

```
label, results, err := Roll("d20+16 | c")
label, result, err := RollOnce("15d6 + 15 fire + 1 acid")
```

If you need to keep the die roller itself around after the dice are rolled, to query its status, or to produce a repeatable string of die rolls given a custom seed or number generator, create a new DieRoller value and reuse that as needed:

```
dr, err := NewDieRoller()
label, results, err := dr.DoRoll("d20+16 | c")
```

“Factored” Notation

```
import "fmt"  
import "encoding/json"  
import "flag"  
import "math"
```

“Factored” Notation

```
import "fmt"
import "encoding/json"
import "flag"
import "math"

import (
    "fmt"
    "encoding/json"
    "flag"
    "math"
)
```

“Factored” Notation

```
var initialized bool
var usernames    []string
var Greeting     string    = "Hello"
var TheAnswer    = 42

var (
    initialized bool
    usernames    []string
    Greeting     string    = "Hello"
    TheAnswer    = 42
)
```

“Factored” Notation

```
const initialized = false
const Greeting   = "Hello"
const TheAnswer  byte = 42

const (
    initialized      = false
    Greeting         = "Hello"
    TheAnswer        byte = 42
)
```

“Factored” Notation and iota

```
type MessageType byte
const (
    ServerCommand MessageType = 0
    ServerReply    MessageType = 1
    ServerError    MessageType = 2
    UrgentMessage  MessageType = 3
)
```


“Factored” Notation and iota

```
type MessageType byte
const (
    ServerCommand MessageType = 0
    ServerReply    MessageType = 1
    ServerError    MessageType = 2
    UrgentMessage  MessageType = 3
)
```

```
type MessageType byte
const (
    ServerCommand MessageType = iota
    ServerReply    MessageType = iota
    ServerError    MessageType = iota
    UrgentMessage  MessageType = iota
)
```

“Factored” Notation and iota

```
type MessageType byte
const (
    ServerCommand MessageType = 0
    ServerReply      MessageType = 1
    ServerError      MessageType = 2
    UrgentMessage    MessageType = 3
)
```

```
type MessageType byte
const (
    ServerCommand MessageType = iota
    ServerReply
    ServerError
    UrgentMessage
)
```

“Factored” Notation and iota

```
type MessageType byte
const (
    ServerCommand MessageType = 0
    ServerReply      MessageType = 1
    ServerError      MessageType = 2
    UrgentMessage    MessageType = 3
)
```

```
type MessageType byte
const (
    ServerCommand MessageType = 1 << iota
    ServerReply
    ServerError
    UrgentMessage
)
```