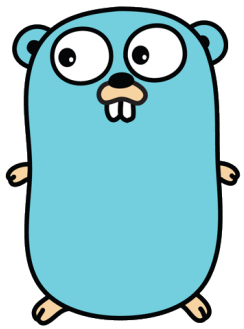
A Whirlwind Tour of Go

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

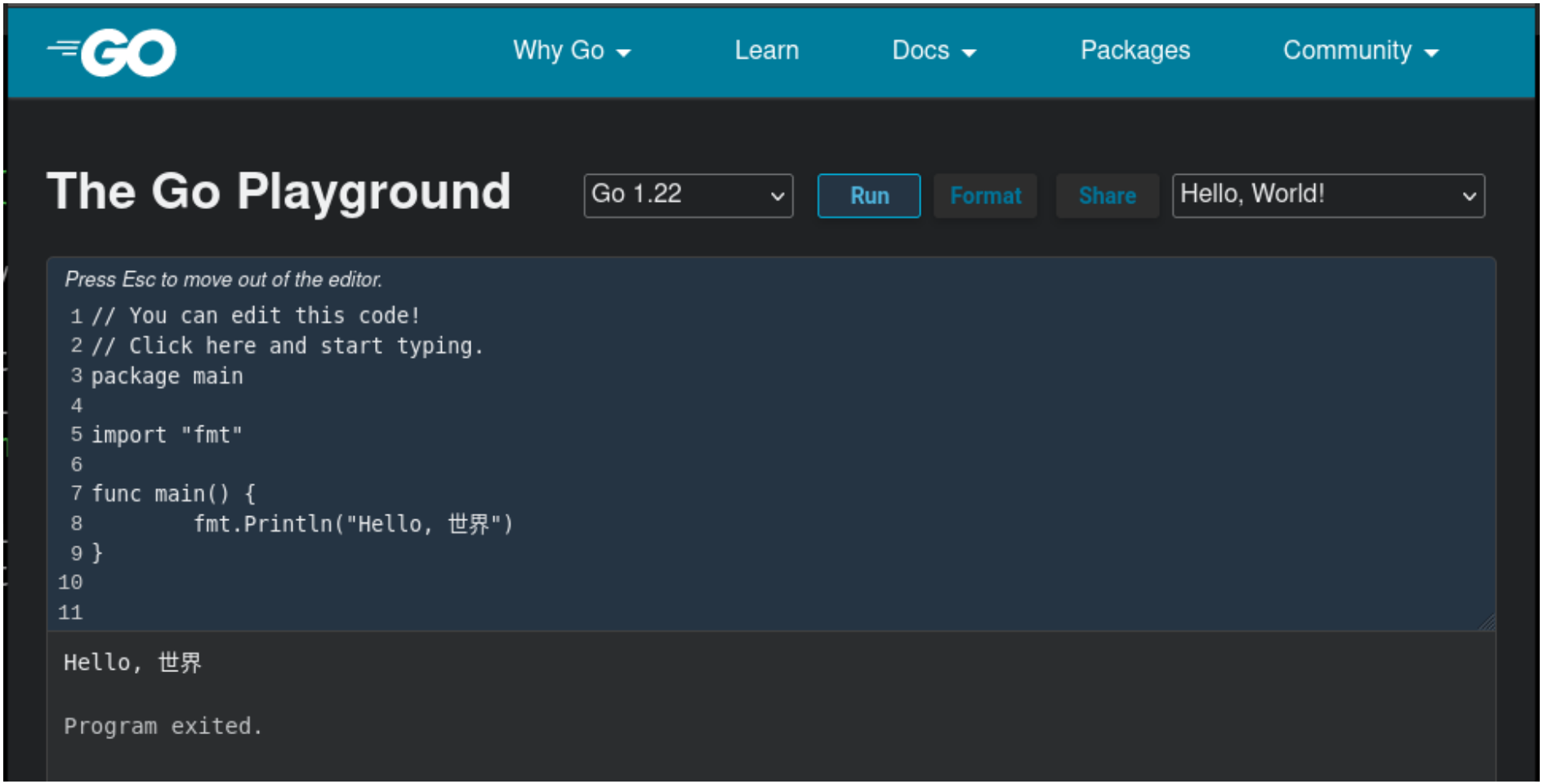
- Basic unit is a *package* (namespace boundary).
- Multiple source files in a package, in the same directory tree.
- Every program must have a `main` package.
- The `main` package has a `main` function.
- Import packages into the program using the `import` statement.
- Always prefix identifiers from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



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:

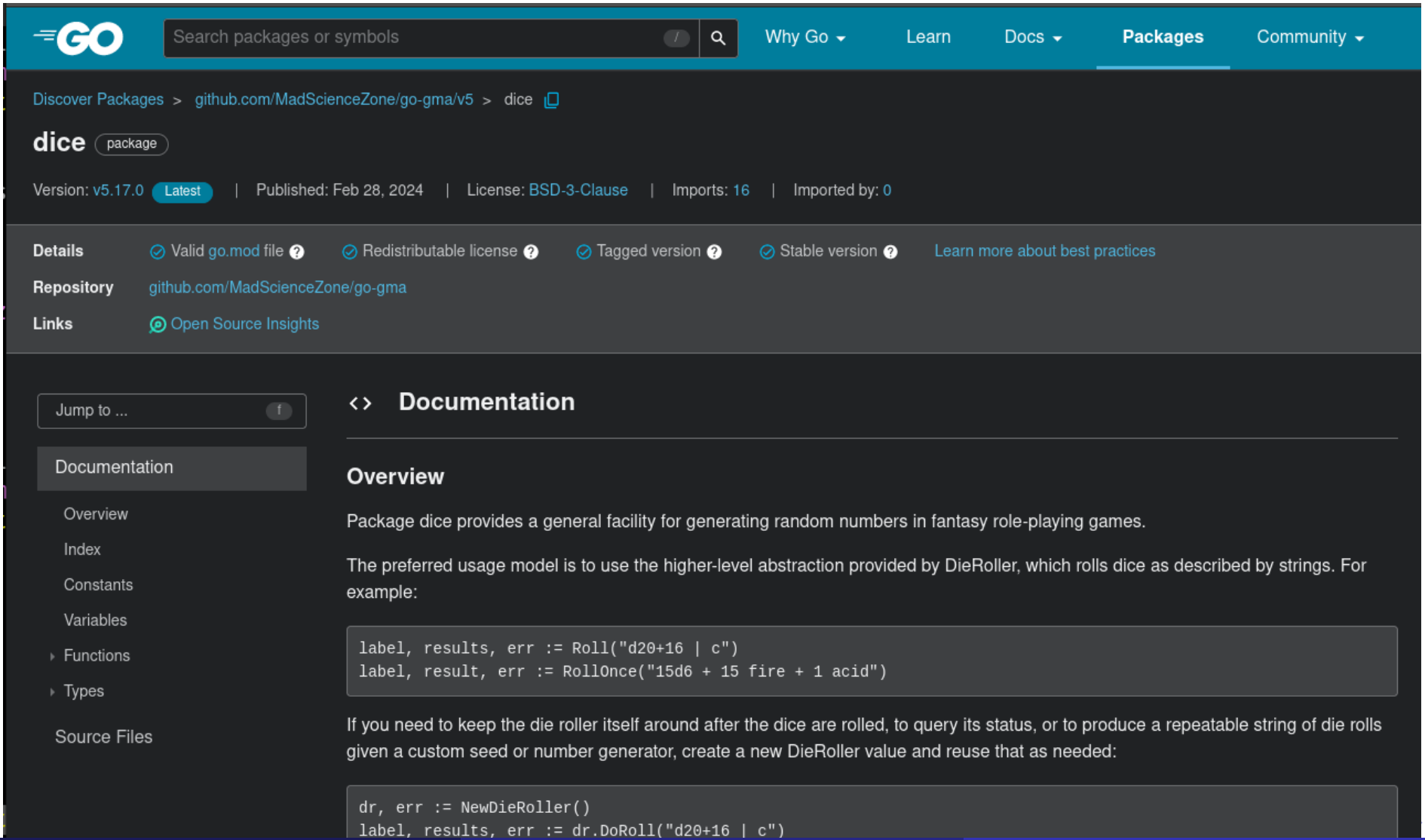
```
import "fmt"
import "encoding/json"
import "flag"
import "math"
```

- Getting packages from public repositories:

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

Automatic API Documentation

- `https://pkg.go.dev/`*repository-url*



“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 Expressions

```
type MessageType byte
const (
    ServerCommand MessageType = 0x01
    ServerReply   MessageType = 0x02
    ServerError   MessageType = 0x04
    UrgentMessage MessageType = 0x08
)
```

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

Global ID Generation (Naïve)

```
type GameState struct {
    NextMessageID int
}
```

Global ID Generation (Naïve)

```
type GameState struct {
    NextMessageID int
}

var gameServer GameState

gameServer.NextMessageID++
client.ID = gameServer.NextMessageID
```

Global ID Generation (Mutex)

```
type GameState struct {
    NextMessageID int
    Lock          sync.Mutex
}
```

Global ID Generation (Mutex)

```
type GameState struct {
    NextMessageID int
    Lock          sync.RWMutex
}
```

Global ID Generation (Mutex)

```
type GameState struct {
    NextMessageID int
    Lock          sync.RWMutex
}

func (state *GameState) GetNextID() int {
    state.Lock.Lock()
    state.NextMessageID++
    nextID := state.MessageID
    state.Lock.Unlock()
    return nextID
}
```

Global ID Generation (Mutex)

```
type GameState struct {
    NextMessageID int
    Lock          sync.RWMutex
}

func (state *GameState) GetNextID() int {
    state.Lock.Lock()
    state.NextMessageID++
    nextID := state.MessageID
    state.Lock.Unlock()
    return nextID
}

client.ID = gameServer.GetNextID()
```

Global ID Generation (Mutex)

```
type GameState struct {
    NextMessageID int
    Lock          sync.RWMutex
}

func (state *GameState) GetNextID() int {
    state.Lock.Lock()
    defer state.Lock.Unlock()

    state.NextMessageID++
    return state.NextMessageID
}

client.ID = gameServer.GetNextID()
```

Global ID Generation (Channel)

```
func serveMessageIDs(c chan int) int {
    var id int
    for {
        c <- id
        c++
    }
}
```

Global ID Generation (Channel)

```
func serveMessageIDs(c chan int) int {
    var id int
    for {
        c <- id
        c++
    }
}

IDSource := make(chan int)
go serveMessageIDs(IDSource)
```

Global ID Generation (Channel)

```
func serveMessageIDs(c chan int) int {
    var id int
    for {
        c <- id
        c++
    }
}

IDSource := make(chan int)
go serveMessageIDs(IDSource)

client.ID = <-IDSource
```

Channels

```
ch := make(chan byte)
```

Channels

```
ch := make(chan byte)

fmt.Println("Writing to channel")

ch <- 42
```

Channels

```
ch := make(chan byte)

fmt.Println("Writing to channel")

ch <- 42

fmt.Println("Reading from channel")
x := <-ch
fmt.Println("Read", x, "from channel")
```

Channels

```
ch := make(chan byte)

fmt.Println("Writing to channel")
ch <- 42          // DEADLOCKED!
fmt.Println("Reading from channel")
x := <-ch
fmt.Println("Read", x, "from channel")
```

Channels

```
ch := make(chan byte)
go func(c chan byte) {
    x := <-c
    fmt.Println("Read", x, "from channel")
}(ch)

fmt.Println("Writing to channel")
ch <- 42
```

Channels

```
ch := make(chan byte, 1)

fmt.Println("Writing to channel")
ch <- 42

fmt.Println("Reading from channel")
x := <-ch
fmt.Println("Read", x, "from channel")
```