

A Whirlwind Tour of Go

Just the Cool Parts

Steve Willoughby

15-Apr-2024

v1.2.0



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...”
 - \therefore 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...”
 - \therefore 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

The Basics

This is the “whirlwind” part...

(Laying a foundation of the basics so the more interesting discussions are understandable.)

Intrinsic Data Types

- The usual suspects: `int`, `int8`, `int16`, `int32`, `int64`, `uint`, `uint8`, `uint16`, `uint32`, `uint64`, `bool`, `byte`, `float32`, `float64`, `string`.
 - `"hello"`, ``hello``, `42`, `0x7F`, `0o177`, `0b11010111`

Intrinsic Data Types

- The usual suspects: `int`, `int8`, `int16`, `int32`, `int64`, `uint`, `uint8`, `uint16`, `uint32`, `uint64`, `bool`, `byte`, `float32`, `float64`, `string`.
 - `"hello"`, ``hello``, 42, 0x7F, 0o177, 0b11010111
- Special things: `complex64`, `complex128`, `error`.
 - 13+2i, `complex`(13, 2)

Intrinsic Data Types

- The usual suspects: `int`, `int8`, `int16`, `int32`, `int64`, `uint`, `uint8`, `uint16`, `uint32`, `uint64`, `bool`, `byte`, `float32`, `float64`, `string`.
 - `"hello"`, ``hello``, 42, 0x7F, 0o177, 0b11010111
- Special things: `complex64`, `complex128`, `error`.
 - 13+2i, `complex`(13, 2)
- Structures: `struct`{...}.

Intrinsic Data Types

- The usual suspects: `int`, `int8`, `int16`, `int32`, `int64`, `uint`, `uint8`, `uint16`, `uint32`, `uint64`, `bool`, `byte`, `float32`, `float64`, `string`.
 - `"hello"`, ``hello``, `42`, `0x7F`, `0o177`, `0b11010111`
- Special things: `complex64`, `complex128`, `error`.
 - `13+2i`, `complex(13, 2)`
- Structures: `struct{...}`.
- What about `char`? Nope. Instead, we have `rune`.
 - `'A'`, `'\n'`, `'Σ'`

Intrinsic Data Types

- The usual suspects: `int`, `int8`, `int16`, `int32`, `int64`, `uint`, `uint8`, `uint16`, `uint32`, `uint64`, `bool`, `byte`, `float32`, `float64`, `string`.
 - `"hello"`, ``hello``, 42, `0x7F`, `0o177`, `0b11010111`
- Special things: `complex64`, `complex128`, `error`.
 - `13+2i`, `complex(13, 2)`
- Structures: `struct{...}`.
- What about `char`? Nope. Instead, we have `rune`.
 - `'A'`, `'\n'`, `'Σ'`
- Arrays: `[10]int`, `[100]rune`.

Intrinsic Data Types

- The usual suspects: `int`, `int8`, `int16`, `int32`, `int64`, `uint`, `uint8`, `uint16`, `uint32`, `uint64`, `bool`, `byte`, `float32`, `float64`, `string`.
 - `"hello"`, ``hello``, 42, 0x7F, 0o177, 0b11010111
- Special things: `complex64`, `complex128`, `error`.
 - 13+2i, `complex`(13, 2)
- Structures: `struct{...}`.
- What about `char`? Nope. Instead, we have `rune`.
 - `'A'`, `'\n'`, `'Σ'`
- Arrays: `[10]int`, `[100]rune`.
- Slices: `[]int`, `[]byte`, `[]string`.

Intrinsic Data Types

- The usual suspects: `int`, `int8`, `int16`, `int32`, `int64`, `uint`, `uint8`, `uint16`, `uint32`, `uint64`, `bool`, `byte`, `float32`, `float64`, `string`.
 - `"hello"`, ``hello``, 42, `0x7F`, `0o177`, `0b11010111`
- Special things: `complex64`, `complex128`, `error`.
 - `13+2i`, `complex(13, 2)`
- Structures: `struct{...}`.
- What about `char`? Nope. Instead, we have `rune`.
 - `'A'`, `'\n'`, `'Σ'`
- Arrays: `[10]int`, `[100]rune`.
- Slices: `[]int`, `[]byte`, `[]string`.
- Maps: `map[string]int`.

Intrinsic Data Types

- The usual suspects: `int`, `int8`, `int16`, `int32`, `int64`, `uint`, `uint8`, `uint16`, `uint32`, `uint64`, `bool`, `byte`, `float32`, `float64`, `string`.
 - `"hello"`, ``hello``, 42, 0x7F, 0o177, 0b11010111
- Special things: `complex64`, `complex128`, `error`.
 - 13+2i, `complex`(13, 2)
- Structures: `struct{...}`.
- What about `char`? Nope. Instead, we have `rune`.
 - `'A'`, `'\n'`, `'Σ'`
- Arrays: `[10]int`, `[100]rune`.
- Slices: `[]int`, `[]byte`, `[]string`.
- Maps: `map[string]int`.
- Channels: `chan int`.

Expressions and Operators

- Arithmetic: $+$, $-$, $*$, $/$, $\%$.
- Relational: $==$, $!=$, $>$, $<$, $>=$, $<=$.
- Logical: $\&\&$, $||$, $!$.
- Bitwise: $\&$, $|$, \wedge , $<<$, $>>$, $\&\wedge$.

// $x \&\wedge y == x \& (\wedge y)$

Expressions and Operators

- Arithmetic: `+`, `-`, `*`, `/`, `%`.
- Relational: `==`, `!=`, `>`, `<`, `>=`, `<=`.
- Logical: `&&`, `||`, `!`.
- Bitwise: `&`, `|`, `^`, `<<`, `>>`, `&^`.
`// x &^ y == x & (^y)`
- Assignment: `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `^=`, `|=`, `<<=`, `>>=`, `:=`.

Expressions and Operators

- Arithmetic: `+`, `-`, `*`, `/`, `%`.
- Relational: `==`, `!=`, `>`, `<`, `>=`, `<=`.
- Logical: `&&`, `||`, `!`.
- Bitwise: `&`, `|`, `^`, `<<`, `>>`, `&^`. `// x &^ y == x & (^y)`
- Assignment: `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `^=`, `|=`, `<<=`, `>>=`, `:=`.
- Reference/Dereference: `&`, `*`.
- Unary: `+`, `-`, `^`. `// ^x`
- Increment/Decrement: `++`, `--`. `// x++ or x--`

Expressions and Operators

- Arithmetic: `+`, `-`, `*`, `/`, `%`.
- Relational: `==`, `!=`, `>`, `<`, `>=`, `<=`.
- Logical: `&&`, `||`, `!`.
- Bitwise: `&`, `|`, `^`, `<<`, `>>`, `&^`. `// x &^ y == x & (^y)`
- Assignment: `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `^=`, `|=`, `<<=`, `>>=`, `:=`.
- Reference/Dereference: `&`, `*`.
- Unary: `+`, `-`, `^`. `// ^x`
- Increment/Decrement: `++`, `--`. `// x++ or x--`
- Channel I/O: `<-`. `// channel<-x or <-channel`

Expressions and Operators

- Arithmetic: `+`, `-`, `*`, `/`, `%`.
- Relational: `==`, `!=`, `>`, `<`, `>=`, `<=`.
- Logical: `&&`, `||`, `!`.
- Bitwise: `&`, `|`, `^`, `<<`, `>>`, `&^`. `// x &^ y == x & (^y)`
- Assignment: `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `^=`, `|=`, `<<=`, `>>=`, `:=`.
- Reference/Dereference: `&`, `*`.
- Unary: `+`, `-`, `^`. `// ^x`
- Increment/Decrement: `++`, `--`. `// x++ or x--`
- Channel I/O: `<-`. `// channel<-x or <-channel`
- Blank identifier: `_`.

Declarations

- Type declarations *follow* identifier names

```
var x int
```

```
var UserName string
```

Declarations

- Type declarations *follow* identifier names

```
var x int
```

```
var UserName string
```

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

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

Declarations

- 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.

Program Structure

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

```
https://go.dev/play/
```

```
/* Standard-issue "Hello, World" program in Go */
```

```
package main
```

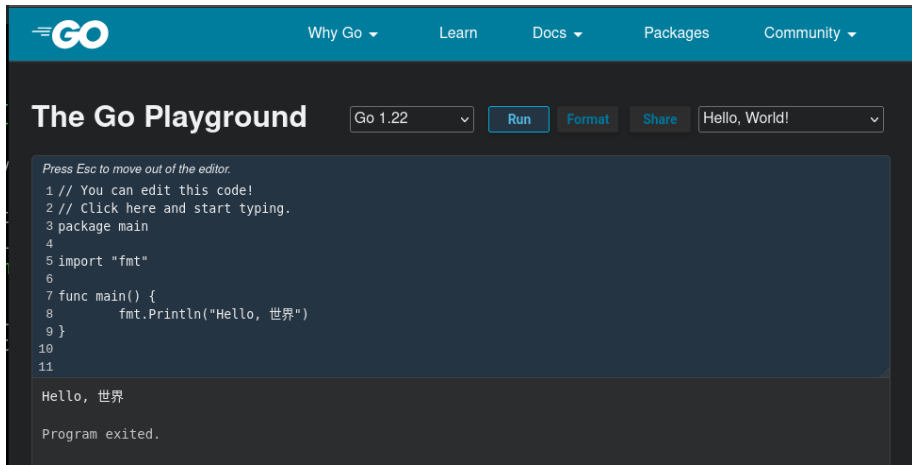
```
import "fmt"
```

```
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 header area with the title 'The Go Playground' and controls for the Go version (set to 1.22), buttons for 'Run', 'Format', and 'Share', and a dropdown menu showing the current output 'Hello, World!'. The main area is a code editor with a dark background. It contains a Go program that prints 'Hello, 世界'. Below the editor, the output shows 'Hello, 世界' and 'Program exited.'.

Press Esc to move out of the editor.

```
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
```

Hello, 世界

Program exited.

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 header includes the Go logo, a search bar, and navigation links like 'Why Go', 'Learn', 'Docs', 'Packages', and 'Community'. The main content area displays the package name 'dice' with a 'package' tag, its version 'v5.17.0' (marked as 'Latest'), and publication details. Below this, there are sections for 'Details' (with links to go.mod, license, tagged version, and stable version), 'Repository' (github.com/MadScienceZone/go-gma), and 'Links' (Open Source Insights). The 'Documentation' section is expanded, showing a sidebar with 'Overview', 'Index', 'Constants', 'Variables', 'Functions', 'Types', and 'Source Files'. The 'Overview' section describes the package's purpose and provides a code example for rolling dice.

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](#)

<> Documentation

Documentation

- Overview
- Index
- Constants
- Variables
- Functions
- Types
- Source Files

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()
```

“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

<https://go.dev/play/p/LSHu1VKUz20>

<https://tinyurl.com/gofactor>

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



“Factored” Notation and iota

<https://go.dev/play/p/LSHu1VKUz20>

<https://tinyurl.com/gofactor>

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

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



“Factored” Notation and iota

<https://go.dev/play/p/LSHu1VKUz20>

<https://tinyurl.com/gofactor>

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

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



“Factored” Notation and iota Expressions

<https://go.dev/play/p/LSHu1VKUz20>

<https://tinyurl.com/gofactor>

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

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



Conditionals

```
var x int

if x > 10 {
    fmt.Println("X exceeds 10.")
} else {
    fmt.Println("X is tiny.")
}
```

Conditionals

```
var x int

if x > 10 {
    fmt.Println("X exceeds 10.")
} else {
    fmt.Println("X is tiny.")
}

if x *= 2; x > 10 {
    fmt.Println("Now X is big.")
} else {
    fmt.Println("X is still small.")
}
```

Switches

```
var x int

switch x {
case 0:
    fmt.Println("X is nothing.")
case 1, 3, 5:
    fmt.Println("X is odd.")
case 2, 4, 6:
    fmt.Println("X is even.")
default:
    fmt.Println("X is bigger than I can count.")
}
```

Loops

```
// infinite loop
```

```
for {  
}
```

```
// while loop
```

```
for thing.IsReady() {  
}
```

```
// traditional 3-part for loop
```

```
for i := 0; i < 10; i++ {  
}
```

Loops

```
// loop over interval [0,10)
```

```
for i := range 10 {  
}
```

```
// loop over elements of a collection
```

```
for i, v := range []int{1, 4, -3, 153} {  
}
```

```
for key, val := range map[int]int{1:2, 3:4, ...} {  
}
```

```
// loop over data received from channel
```

```
for item := range channel {  
}
```


Arrays

- The number of elements *is part of the type* (`[10] int` vs. `[15] int`).

Arrays

- The number of elements *is part of the type* ([10] `int` vs. [15] `int`).
- Variables declared are initialized empty but ready for use

```
var things [5]string
```

```
things[0] = "raindrops on roses"  
things[1] = "whiskers on kittens"  
things[2] = "copper kettles"  
things[3] = "woolen mittens"  
things[4] = "wild geese"
```

```
fmt.Println("I like", things[2])  
fmt.Println("I also like", things)  
fmt.Println("I know", len(things), "things.")
```

Arrays

<https://go.dev/play/p/rexZjp6SdKD>

<https://tinyurl.com/goarrays>

- Or you can specify an array literal value to use in an expression or assign to a variable

```
things := [5]string{  
    "raindrops on roses",  
    "whiskers on kittens",  
    "copper kettles",  
    "woolen mittens",  
    "wild geese",  
}
```



```
fmt.Println("I like", things[2])  
fmt.Println("I also like", things)  
fmt.Println("I know", len(things), "things.")
```

Slices

<https://go.dev/play/p/rexZjp6SdKD>

<https://tinyurl.com/goarrays>

- Specify a range $[n:m]$ as the index into an array to get a subset of the array values with indices from n to $m - 1$.
- The value is a *slice*, not an *array*. It's a different type.
 - For `[5]string`, the value is `[]string`.

```
fmt.Println("Some things:", things[1:3])
fmt.Println("Some things:", things[:3])
fmt.Println("Some things:", things[1:])
fmt.Println("Some things:", things[:])
```



Slices

<https://go.dev/play/p/rexZjp6SdKD>

<https://tinyurl.com/goarrays>

- Dimensionless “arrays”: `[]int`.
- Actually a “view” into an underlying array.
 - Go creates and manages the underlying array automatically for you.

```
var things []string
```



Slices

<https://go.dev/play/p/rexZjp6SdKD>

<https://tinyurl.com/goarrays>

- Dimensionless “arrays”: `[]int`.
- Actually a “view” into an underlying array.
 - Go creates and manages the underlying array automatically for you.

```
var things []string
```

```
things = append(things, "doorbells")
```

```
things = append(things, "sleighbells", "schnitzel")
```

```
fmt.Println(len(things), things)
```

```
// prints: 3 [doorbells sleighbells schnitzel]
```



Slices

<https://go.dev/play/p/rexZjp6SdKD>

<https://tinyurl.com/goarrays>

- Can also specify a slice of values as a literal.

```
things := []string{  
    "doorbells",  
    "sleighbells",  
    "schnitzel",  
}
```



Slices

<https://go.dev/play/p/rexZjp6SdKD>

<https://tinyurl.com/goarrays>

- Can also specify a slice of values as a literal.

```
things := []string{
    "doorbells",
    "sleighbells",
    "schnitzel",
}
```



```
primes := []int{2, 3, 5, 7, 11, 13}
lowPrimes := slices.Delete(primes, 3, len(primes))
fmt.Println(lowPrimes)
// prints: [2 3 5]
```


Maps

<https://go.dev/play/p/Bfs6kEUKwve>

<https://tinyurl.com/go-kvmap>

```
var Ages map[string]int
Ages = make(map[string]int)
```



Maps

<https://go.dev/play/p/Bfs6kEUKwve>

<https://tinyurl.com/go-kvmap>

```
var Ages map[string]int
Ages = make(map[string]int)

Ages["Alice"] = 14
Ages["Bob"] = 22
Ages["Charlie"] = 27
Ages["Daria"] = 42
fmt.Println(Ages)
```



Maps

<https://go.dev/play/p/Bfs6kEUKwve>

<https://tinyurl.com/go-kvmap>

```
Ages := map[string]int{
    "Alice": 14,
    "Bob": 22,
    "Charlie": 27,
    "Daria": 42,
}

fmt.Println(Ages)

for name, age := range Ages {
    if age >= 18 {
        fmt.Printf("%s may vote.\n", name)
    } else {
        fmt.Printf("%s is not eligible.\n", name)
    }
}
```



Maps

```
https://go.dev/play/p/Bfs6kEUKwve
```

```
aliceAge := Ages["Alice"]  
eveAge  := Ages["Eve"]
```

```
https://tinyurl.com/go-kvmap
```

```
// 14  
// 0
```



Maps

```
https://go.dev/play/p/Bfs6kEUKwve
```

```
aliceAge := Ages["Alice"]  
eveAge  := Ages["Eve"]
```

```
https://tinyurl.com/go-kvmap
```

```
// 14  
// 0
```

```
aliceAge, exists := Ages["Alice"]  
eveAge, exists  := Ages["Eve"]
```

```
// 14, true  
// 0, false
```



Maps

```
https://go.dev/play/p/Bfs6kEUKwve
```

```
https://tinyurl.com/go-kvmap
```

```
aliceAge := Ages["Alice"] // 14
eveAge  := Ages["Eve"]    // 0
```

```
aliceAge, exists := Ages["Alice"] // 14, true
eveAge, exists  := Ages["Eve"]    // 0, false
```

```
Ages["Eve"] = 20
delete(Ages, "Bob")
```



Maps

<https://go.dev/play/p/Bfs6kEUKwve>

<https://tinyurl.com/go-kvmap>

```
aliceAge := Ages["Alice"] // 14
eveAge   := Ages["Eve"]   // 0
```

```
aliceAge, exists := Ages["Alice"] // 14, true
eveAge, exists   := Ages["Eve"]   // 0, false
```

```
Ages["Eve"] = 20
delete(Ages, "Bob")
```

```
if _, exists := Ages[name]; exists {
    fmt.Println("We do know about", name)
}
```



Maps

<https://go.dev/play/p/Bfs6kEUKwve>

<https://tinyurl.com/go-kvmap>

```
aliceAge := Ages["Alice"] // 14
eveAge  := Ages["Eve"]    // 0
```

```
aliceAge, exists := Ages["Alice"] // 14, true
eveAge, exists  := Ages["Eve"]    // 0, false
```

```
Ages["Eve"] = 20
delete(Ages, "Bob")
```

```
if age, exists := Ages[name]; exists {
    fmt.Printf("We know %s's age is %d.\n", name, age)
} else {
    fmt.Println("We don't know", name)
}
```



Error Values

Error Handling

Error Values

```
func main() {  
    var intval int  
    var err      error  
  
    for i, arg := range os.Args[1:] {  
        intval, err = strconv.Atoi(arg)  
        if err != nil {  
            fmt.Printf("Arg #%d (\"%s\"): %v.\n",  
                        i, arg, err)  
        } else {  
            fmt.Printf("Arg #%d == %d\n", i, intval)  
        }  
    }  
}
```

Object Oriented Features

Wherewith Object Orientation?

(For a language without object classes...)

Structures

<https://go.dev/play/p/TC1HUvPovi0>

<https://tinyurl.com/go-obj-oop>

```
type Triangle struct {  
    Base    int  
    Height  int  
    X       int // Reference point  
    Y       int  
}
```



Structures

<https://go.dev/play/p/TC1HUvPovi0>

<https://tinyurl.com/go-obj-oop>

```
type Triangle struct {  
    Base    int  
    Height  int  
    X       int // Reference point  
    Y       int  
}
```

```
var t1 Triangle  
t1.Base = 37  
t1.Height = 15  
t1.X = 11  
t1.Y = 22
```



Structures

<https://go.dev/play/p/TC1HUvPovi0>

<https://tinyurl.com/go-obj-oop>

```
type Triangle struct {  
    Base    int  
    Height  int  
    X       int // Reference point  
    Y       int  
}  
  
var t2 Triangle = Triangle{Base: 3, Height: 1}  
  
fmt.Println("t2's base is", t2.Base)
```



Structures

<https://go.dev/play/p/TC1HUvPovi0>

<https://tinyurl.com/go-obj-oop>

```
type Triangle struct {  
    Base    int  
    Height  int  
    X       int // Reference point  
    Y       int  
}  
  
t3 := Triangle{  
    Base:    100,  
    Height:  42,  
    X:       -3,  
    Y:       14,  
}
```



Method Functions

<https://go.dev/play/p/TC1HUvPovi0>

<https://tinyurl.com/go-obj-oop>

```
func Area(t Triangle) float64 {  
    return (float64(t.Base) *  
           float64(t.Height)) / 2.0  
}  
  
func Translate(t Triangle, dx, dy int) Triangle {  
    t.X += dx  
    t.Y += dy  
    return t  
}  
  
fmt.Println("t1 area =", Area(t1))  
t2 = Translate(t2, +3, -2)
```



Method Functions

<https://go.dev/play/p/TC1HUvPovi0><https://tinyurl.com/go-obj-oop>

```
func Area(t Triangle) float64 {  
    return (float64(t.Base) *  
        float64(t.Height)) / 2.0  
}  
  
func Translate(t *Triangle, dx, dy int) {  
    t.X += dx  
    t.Y += dy  
}  
  
fmt.Println("t1 area =", Area(t1))  
Translate(&t2, +3, -2)
```



Method Functions

<https://go.dev/play/p/TC1HUvPovi0><https://tinyurl.com/go-obj-oop>

```
func (t Triangle) Area() float64 {  
    return (float64(t.Base) *  
        float64(t.Height)) / 2.0  
}  
  
func (t *Triangle) Translate(dx, dy int) {  
    t.X += dx  
    t.Y += dy  
}  
  
fmt.Println("t1 area =", t1.Area())  
t2.Translate(+3, -2)
```



Composition

<https://go.dev/play/p/qp2nc6gywLr>

<https://tinyurl.com/go-obj-comp>

```
type BaseShape struct {  
    X int  
    Y int  
}  
  
func (s BaseShape) ReferencePoint() (int, int) {  
    return s.X, s.Y  
}  
  
// (We'll set aside the Translate method for now to  
// keep the in-class example simple.)
```



Composition

<https://go.dev/play/p/qp2nc6gywLr>

<https://tinyurl.com/go-obj-comp>

```
type Triangle struct {  
    BaseShape  
    Base    int  
    Height  int  
}  
  
func (t Triangle) Area() float64 {  
    return (float64(t.Base) *  
            float64(t.Height)) / 2.0  
}
```



Composition

<https://go.dev/play/p/qp2nc6gywLr>

<https://tinyurl.com/go-obj-comp>

```
type Rectangle struct {  
    BaseShape  
    Width    int  
    Height   int  
}  
  
func (r Rectangle) Area() float64 {  
    return float64(r.Width * r.Height)  
}
```



Composition

<https://go.dev/play/p/qp2nc6gywLr>

<https://tinyurl.com/go-obj-comp>

```
// Regular Polygons
type Polygon struct {
    BaseShape
    Sides    int
    Length   float64 // Length of each side
    Radius   float64 // Radius of inscribed circle
}

func (p Polygon) Area() float64 {
    return (float64(p.Sides) / 2.0) *
        p.Length * p.Radius
}
```



Composition

<https://go.dev/play/p/qp2nc6gywLr>

<https://tinyurl.com/go-obj-comp>

```
type Circle struct {  
    BaseShape  
    Radius float64  
}  
  
func (c Circle) Area() float64 {  
    return math.Pi * math.Pow(c.Radius, 2)  
}
```



Composition

<https://go.dev/play/p/qp2nc6gywLr>

<https://tinyurl.com/go-obj-comp>

```
c := Circle{  
    BaseShape: BaseShape{  
        X: 1,  
        Y: 2,  
    },  
    Radius: 1.5,  
}
```



Composition

<https://go.dev/play/p/qp2nc6gywLr>

<https://tinyurl.com/go-obj-comp>

```
c := Circle{
    BaseShape: BaseShape{
        X: 1,
        Y: 2,
    },
    Radius: 1.5,
}

fmt.Println(c.BaseShape.X, c.BaseShape.Y,
            c.Radius, c.Area())
```



Composition

<https://go.dev/play/p/qp2nc6gywLr>

<https://tinyurl.com/go-obj-comp>

```
c := Circle{
    BaseShape: BaseShape{
        X: 1,
        Y: 2,
    },
    Radius: 1.5,
}

fmt.Println(c.BaseShape.X, c.BaseShape.Y,
            c.Radius, c.Area())

fmt.Println(c.X, c.Y, c.Radius, c.Area())
```



Polymorphism

<https://go.dev/play/p/qp2nc6gywLr>

<https://tinyurl.com/go-obj-comp>

```
shapes := []Shape{
    Triangle{
        BaseShape: BaseShape{
            X: 3,
            Y: 12,
        },
        Base: 3,
        Height: 2,
    },
    Circle{BaseShape:BaseShape{Y: 22}, Radius: 1.5},
    Rectangle{Height: 100, Width: 50},
}
```



Polymorphism

<https://go.dev/play/p/qp2nc6gywLr>

<https://tinyurl.com/go-obj-comp>

```
shapes := []Shape{
    Triangle{BaseShape: BaseShape{X: 3, Y: 12},
        Base: 3, Height: 2},
    Circle{BaseShape: BaseShape{Y: 22}, Radius: 1.5},
    Rectangle{Height: 100, Width: 50},
}

for i, shape := range shapes {
    x, y := shape.ReferencePoint()
    fmt.Printf("#%d at (%d,%d), area=%f\n",
        i, x, y, shape.Area())
}
```



Polymorphism via Interfaces

<https://go.dev/play/p/qp2nc6gywLr>

<https://tinyurl.com/go-obj-comp>

```
type Shape interface {  
    Area() float64  
    ReferencePoint() (int, int)  
}
```



Polymorphism via Interfaces

<https://go.dev/play/p/qp2nc6gywLr>

<https://tinyurl.com/go-obj-comp>

```
type Shape interface {  
    Area() float64  
    ReferencePoint() (int, int)  
}  
  
func reportArea(s Shape) {  
    fmt.Printf("The area is %f\n", s.Area())  
}
```



Type Assertions

Type Assertions

Type Assertions

<https://go.dev/play/p/dy952C3yZUX>

<https://tinyurl.com/go-assert>

```
f(42)
```

```
f(-2)
```

```
func f(mystery any) {    // any == interface{}
    var v int

    // we know it's an int, just treat it as one
    v = mystery + 15

    fmt.Println("int mystery is", v)
}
```



Type Assertions

<https://go.dev/play/p/dy952C3yZUX>

<https://tinyurl.com/go-assert>

```
f(42)
```

```
f(-2)
```

```
func f(mystery any) {    // any == interface{}
    var v int

    x := mystery.(int)
    v = x + 15

    fmt.Println("int mystery is", v)
}
```



Type Assertions

<https://go.dev/play/p/dy952C3yZUX>

<https://tinyurl.com/go-assert>

```
f(42)
```

```
f("hello")
```

```
func f(mystery any) {    // any == interface{}
    var v int

    x := mystery.(int)
    v = x + 15

    fmt.Println("int mystery is", v)
}
```



Type Assertions

<https://go.dev/play/p/dy952C3yZUX>

<https://tinyurl.com/go-assert>

```
f(42)
f("hello")

func f(mystery any) {    // any == interface{}
    var v int

    x, ok := mystery.(int)
    v = x + 15

    fmt.Println("int mystery is", v)
}
```



Type Switch

<https://go.dev/play/p/dy952C3yZUX><https://tinyurl.com/go-assert>

```
f(42)
f("hello")
func f(mystery any) {    // any == interface{}
    var v int

    switch x := mystery.(type) {
    case int:
        v = x + 15
    case string:
        fmt.Println("string", x)
    default:
        // handle the unknown type
    }
}
```



Goroutines

Concurrency!

(Goroutines and Channels and some other things)

Goroutines—Calling a Function in the “Background”

<https://go.dev/play/p/FJNb0cNYI8->

<https://tinyurl.com/go-routine>

```
func countdown() {  
    for i := 10; i >= 0; i-- {  
        fmt.Printf(">>> %d <<<\n", i)  
        time.Sleep(1 * time.Second)  
    }  
}
```



Goroutines—Calling a Function in the “Background”

<https://go.dev/play/p/FJNb0cNYI8->

<https://tinyurl.com/go-routine>

```
func countdown() {  
    for i := 10; i >= 0; i-- {  
        fmt.Printf(">>> %d <<<\n", i)  
        time.Sleep(1 * time.Second)  
    }  
}  
  
func main() {  
    countdown()  
    fmt.Println("Starting a long-running task...")  
    time.Sleep(15 * time.Second)  
    fmt.Println("Done. Exiting.")  
}
```



Goroutines—Calling a Function in the “Background”

<https://go.dev/play/p/FJNb0cNYI8->

<https://tinyurl.com/go-routine>

```
func countdown() {  
    for i := 10; i >= 0; i-- {  
        fmt.Printf(">>> %d <<<\n", i)  
        time.Sleep(1 * time.Second)  
    }  
}  
  
func main() {  
    go countdown()  
    fmt.Println("Starting a long-running task...")  
    time.Sleep(15 * time.Second)  
    fmt.Println("Done. Exiting.")  
}
```



Channels

<https://go.dev/play/p/sFuh0uwVS6c>

<https://tinyurl.com/go-channel>

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



Channels

<https://go.dev/play/p/sFuh0uwVS6c>

<https://tinyurl.com/go-channel>

```
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

<https://go.dev/play/p/sFuh0uwVS6c>

<https://tinyurl.com/go-channel>

```
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

<https://go.dev/play/p/sFuh0uwVS6c>

<https://tinyurl.com/go-channel>

```
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
```



Buffered Channels

<https://go.dev/play/p/sFuh0uwVS6c>

<https://tinyurl.com/go-channel>

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



Buffered Channels

<https://go.dev/play/p/sFuh0uwVS6c>

<https://tinyurl.com/go-channel>

```
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")
```



Select

Select

Select (C)

```
#include <sys/select.h>
fd_set read_handles, write_handles, err_handles;
struct timeval t;
int sel = 0;

FD_ZERO(&read_handles);
FD_ZERO(&write_handles);
FD_ZERO(&err_handles);

/* set bits for handles you're interested in */
t.tv_sec = 1;
t.tv_usec = 0;
```


Select (C)

```
if ((sel = select(maxfd, &read_handles,  
                  &write_handles,  
                  &err_handles, &t)) == -1) {  
    /* error... */  
}  
else if (sel > 0) {  
    /* go back and check the bits in *_handles */  
}
```

Select (Go)

```
select {  
case x := <- ichan:  
    // we could read from ichan, proceed  
    // with that...  
  
case ochan <- ovalue:  
    // we could write to ochan  
  
}
```

Select (Go)

```
select {  
case x := <- ichan:  
    // we could read from ichan, proceed  
    // with that...  
  
case ochan <- ovalue:  
    // we could write to ochan  
  
default:  
    // if we don't want the whole select to  
    // block, add a default case here.  
  
}
```

Global ID Generation Examples

Thread-Safe Memory Access

by example

Global ID Generation (Naïve)

https://go.dev/play/p/i0xsFX_TSaa

<https://tinyurl.com/go-idgen>

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



Global ID Generation (Naïve)

https://go.dev/play/p/i0xsFX_TSaa

<https://tinyurl.com/go-idgen>

```
type GameState struct {  
    NextMessageID int  
}  
  
var gameserver GameState  
  
// In many concurrent goroutines...  
  
gameServer.NextMessageID++  
client.ID = gameServer.NextMessageID
```



Global ID Generation (Naïve)

https://go.dev/play/p/iOxsFX_TSaa

<https://tinyurl.com/go-idgen>

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

```
var gameserver GameState
```

```
// In many concurrent goroutines...
```

```
gameServer.NextMessageID++           // UNSAFE!
```

```
client.ID = gameServer.NextMessageID // UNSAFE!
```



Global ID Generation (Mutex)

https://go.dev/play/p/i0xsFX_TSaa

<https://tinyurl.com/go-idgen>

```
type GameState struct {
    nextMessageID int
    lock          sync.Mutex
}

func (state *GameState) GetNextID() int {

}

// in many random goroutines...
client.ID = gameServer.GetNextID()
```



Global ID Generation (Mutex)

https://go.dev/play/p/i0xsFX_TSaa

<https://tinyurl.com/go-idgen>

```
type GameState struct {
    nextMessageID int
    lock          sync.Mutex
}

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

// in many random goroutines...
client.ID = gameServer.GetNextID()
```



Global ID Generation (Mutex)

https://go.dev/play/p/i0xsFX_TSaa

<https://tinyurl.com/go-idgen>

```
type GameState struct {
    nextMessageID int
    lock          sync.Mutex
}

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

    state.nextMessageID++
    return state.nextMessageID
}

// in many random goroutines...
client.ID = gameServer.GetNextID()
```



Global ID Generation (Mutex)

https://go.dev/play/p/i0xsFX_TSaa

<https://tinyurl.com/go-idgen>

```
func main() {  
    var gameServer GameState  
    var wg          sync.WaitGroup  
    for i := range 100 {  
        wg.Add(1)  
        id := i  
        go func() {  
            defer wg.Done()  
            fmt.Printf("Goroutine %d, ID=%d\n", id,  
                      gameServer.GetNextID())  
        }()  
    }  
    wg.Wait()  
}
```



Global ID Generation (Channel)

But that's not very idiomatic for
Go.

Here's a much better approach...

Global ID Generation (Channel)

<https://go.dev/play/p/mRfglxbH-kI>

<https://tinyurl.com/go-id-chan>

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



Global ID Generation (Channel)

<https://go.dev/play/p/mRfglxbH-kI>

<https://tinyurl.com/go-id-chan>

```
func serveMessageIDs(c chan<- int) {  
    var id int  
    for {  
        c <- id  
        id++  
    }  
}  
  
// start up the service  
IDSource := make(chan int)  
go serveMessageIDs(IDSource)
```



Global ID Generation (Channel)

<https://go.dev/play/p/mRfglxbH-kI>

<https://tinyurl.com/go-id-chan>

```
func serveMessageIDs(c chan<- int) {  
    var id int  
    for {  
        c <- id  
        id++  
    }  
}  
  
// start up the service  
IDSource := make(chan int)  
go serveMessageIDs(IDSource)  
  
// In many random goroutines...  
client.ID = <-IDSource
```



Bonus/Backup Material

Some more cool stuff if time
allows...

Contextx

Contexts

Contexts

<https://go.dev/play/p/cPSNhVfS8r-><https://tinyurl.com/go-context>

```
func collectData(stream <-chan string) error {
    for {
        data, ok := <-stream
        if !ok {
            return nil
        }
        if err := doSomething(data); err != nil {
            return err
        }
    }
}

// elsewhere
collectData(stream)
```



Contexts

<https://go.dev/play/p/cPSNhVfS8r-><https://tinyurl.com/go-context>

```
func collectData(ctx context.Context,
                 stream <-chan string) error {
    for {
        select {
        case <-ctx.Done():
            return nil
        case data, ok := <-stream:
            if !ok { return nil }
            if err := doSomething(data); err != nil {
                return err
            }
        }
    }
}
```



Contexts

<https://go.dev/play/p/cPSNhVfS8r->

<https://tinyurl.com/go-context>

```
// caller
```

```
ctx, cancel := context.WithTimeout(  
    context.Background(), 5 * time.Second)  
defer cancel()  
  
if err := collectData(ctx, stream); err != nil {  
    panic(err)  
}
```



Contexts

<https://go.dev/play/p/cPSNhVfS8r-><https://tinyurl.com/go-context>

```
func collectData(ctx context.Context,
                stream <-chan string) error {
    for {
        select {
        case <-ctx.Done():
            return nil
        case <-time.After(2 * time.Second):
            log.Print("collectData taking too long")
        case data, ok := <-stream:
            if !ok { return nil }
            if err := doSomething(data); err != nil {
                return err
            }
        }
    }
}
```



JSON

Encoding (JSON)

JSON

https://go.dev/play/p/_KcMkqhzhbZ

<https://tinyurl.com/go-json>

```
import "encoding/json"

type User struct {
    Name    string    `json:"name"`
    ID      int       `json:",omitempty"`
    Attrs   []string  `json:"attributes,omitempty"`
    Secret  []byte    `json:"- "`
}

data := User{
    Name: "steve",
    ID: 42,
    Attrs: []string{"foo", "bar"},
    Secret: sdata,
}
```



JSON

https://go.dev/play/p/_KcMkqhzhbZ

<https://tinyurl.com/go-json>

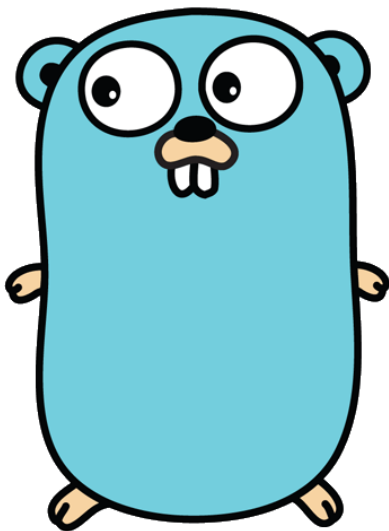
```
import "encoding/json"

type User struct {
    Name    string    `json:"name"`
    ID      int       `json:",omitempty"`
    Attrs   []string  `json:"attributes,omitempty"`
    Secret  []byte    `json:"- "`
}

encoded, err := json.Marshal(data)
// {"name":"steve","ID":42,"attributes":["foo","bar"]}]

var inputData User
err := json.Unmarshal(jsonBytes, &inputData)
```





github.com/MadScienceZone/go-tour