Constant Time

dotGo Paris, 2019

What’s so great about constants?

Immutability

Constants are one of the few ways we have to express immutability in Go

Clarity

Constants give us a way to attribute semantic meant to magic numbers in our code

Performance

Constant folding, constant propogation, branch elimination, dead code elimination

Yeah, so what?

Votre défi

Declare a *constant* whose value is the number of bits in machine word.

```
const uintSize = 32 << (^uint(0) >> 32 & 1)
```

```
const uintSize = 32 << (^uint(0) >> 32 & 1)
```

```
const uintSize = 32 << (^uint(0) >> 32 & 1)
```

```
11111111111111111111111111111111111111111111111111111111111111
```

```
const uintSize = 32 << (^uint(0) >> 32 & 1)
```

```
00000000000000000000000000000000111111111111111111111111111111
```

```
const uintSize = 32 << (^uint(0) >> 32 & 1)
```

```
00000000000000000000000000000001111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000001
```

```
const uintSize = 32 << (^uint(0) >> 32 & 1)
```

```
00000000000000000000000000000000000000000000000000000000000000001
```

```
const uintSize = 32 << (^uint(0) >> 32 & 1)
```

```
const uintSize = 32 << (^uint(0) >> 32 & 1)
```

32 << 1

```
const uintSize = 32 << (^uint(0) >> 32 & 1)
```

64

Constant expressions are evaluated at compile time

In Go, constants are values, and each value has a type

In Go, user defined types can declare their own methods

Thus, a constant value can have a method set

```
const timeout = 500 * time.Millisecond
fmt.Println("The timeout is", timeout) // 500ms
```

```
type Duration int64
    A Duration represents the elapsed time between two instants as an int64
    nanosecond count. The representation limits the largest representable
    duration to approximately 290 years.

func (d Duration) String() string
    String returns a string representing the duration in the form "72h3m0.5s".
    Leading zero units are omitted. As a special case, durations less than one
    second format use a smaller unit (milli-, micro-, or nanoseconds) to
    ensure that the leading digit is non-zero. The zero duration formats as 0s.
```

Constant values can implement interfaces

Singletons

```
package http

// DefaultServeMux is the default ServeMux used by Serve.
var DefaultServeMux = &defaultServeMux
```

```
package http

// DefaultServeMux is the default ServeMux used by Serve.
var DefaultServeMux = &defaultServeMux

var defaultServeMux ServeMux
```

```
package http

// DefaultServeMux is the default ServeMux used by Serve.
var DefaultServeMux = &defaultServeMux

var defaultServeMux ServeMux

// NewServeMux allocates and returns a new ServeMux.
func NewServeMux() *ServeMux { return new(ServeMux) }
```

```
fmt.Fprintf(os.Stdout, "Hello dotGo\n")

syscall.Write(1, []byte("Hello dotGo\n"))
```

```
package os

var (
        Stdin  = NewFile(uintptr(syscall.Stdin), "/dev/stdin")
        Stdout = NewFile(uintptr(syscall.Stdout), "/dev/stdout")
        Stderr = NewFile(uintptr(syscall.Stderr), "/dev/stderr")
)
```

```
type readfd int

func (r readfd) Read(buf []byte) (int, error) {
        return syscall.Read(int(r), buf)}
```

```
type readfd int

func (r readfd) Read(buf []byte) (int, error) {
        return syscall.Read(int(r), buf)}

type writefd int

func (w writefd) Write(buf []byte) (int, error) {
        return syscall.Write(int(w), buf)
}
```

```
type readfd int

func (r readfd) Read(buf []byte) (int, error) {
        return syscall.Read(int(r), buf)}

type writefd int

func (w writefd) Write(buf []byte) (int, error) {
        return syscall.Write(int(w), buf)
}

const (
        Stdin  = readfd(0)
        Stdout = writefd(1)
        Stderr = writefd(2)
)
```

```
type readfd int

func (r readfd) Read(buf []byte) (int, error) {
        return syscall.Read(int(r), buf)}

type writefd int

func (w writefd) Write(buf []byte) (int, error) {
        return syscall.Write(int(w), buf)
}

const (
        Stdin  = readfd(0)
        Stdout = writefd(1)
        Stderr = writefd(2)
)

func main() {
        fmt.Fprintf(Stdout, "Hello world")
}
```

Sentinel error values

```
package io

var EOF = errors.New("EOF")

package sql

var ErrNoRows = errors.New("sql: no rows in result set")

package x509

var ErrUnsupportedAlgorithm = errors.New(
        "x509: cannot verify signature: algorithm unimplemented")
```

```
package nelson

import "io"

func init() {
        io.EOF = nil // haha!
}
```

```
package innocent

import "crypto/rsa"

func init() {
        rsa.ErrVerification = nil
}
```

Sentinal error values behave like singletons, not constants

Fungibility

Fungible means identical

Things which are fungile are by definition equal

```
var myEOF = errors.New("EOF") // io/io.go line 38
```

```
var myEOF = errors.New("EOF") // io/io.go line 38

fmt.Println(myEOF == io.EOF)  // false
```

Constant errors

```
type error interface {
    Error() string
}
```

```
type Error string

func (e Error) Error() string {
        return string(e)
}
```

```
type Error string

func (e Error) Error() string {
        return string(e)
}

const err = Error("EOF")
```

```
type Error string

func (e Error) Error() string {
        return string(e)
}

const err = Error("EOF")

const err2 = errors.errorString{"EOF"}
```

```
type Error string

func (e Error) Error() string {
        return string(e)
}
```

```
type Error string

func (e Error) Error() string {
        return string(e)
}

const err = Error("EOF")
```

```
type Error string

func (e Error) Error() string {
        return string(e)
}

const err = Error("EOF")

err = Error("not EOF")
```

```
const str1 = "EOF"
const str2 = "EOF"
```

```
const str1 = "EOF"
const str2 = "EOF"

fmt.Println(str1 == str2) // true
```

```
const str1 = "EOF"
const str2 = "EOF"

fmt.Println(str1 == str2) // true

type Error string

const err1 = Error("EOF")
const err2 = Error("EOF")
```

```
const str1 = "EOF"
const str2 = "EOF"

fmt.Println(str1 == str2) // true

type Error string

const err1 = Error("EOF")
const err2 = Error("EOF")

fmt.Println(err1 == err2) // true
```

```
% git diff
diff --git a/src/io/io.go b/src/io/io.go
index 2010770e6a..355653b4b8 100644
--- a/src/io/io.go
+++ b/src/io/io.go
@@ -35,7 +35,12 @@ var ErrShortBuffer = errors.New("short buffer")
 // If the EOF occurs unexpectedly in a structured data stream,
 // the appropriate error is either ErrUnexpectedEOF or some other error
 // giving more detail.
-var EOF = errors.New("EOF")
+const EOF = ioError("EOF")
+
+type ioError string
+
+func (e ioError) Error() string { return string(e) }
```

Go’s constants are powerful. If you only think of them as immutable numbers, you’re missing out

Go’s constants let us compose programs that are more correct and harder to misuse

Now, it’s your turn