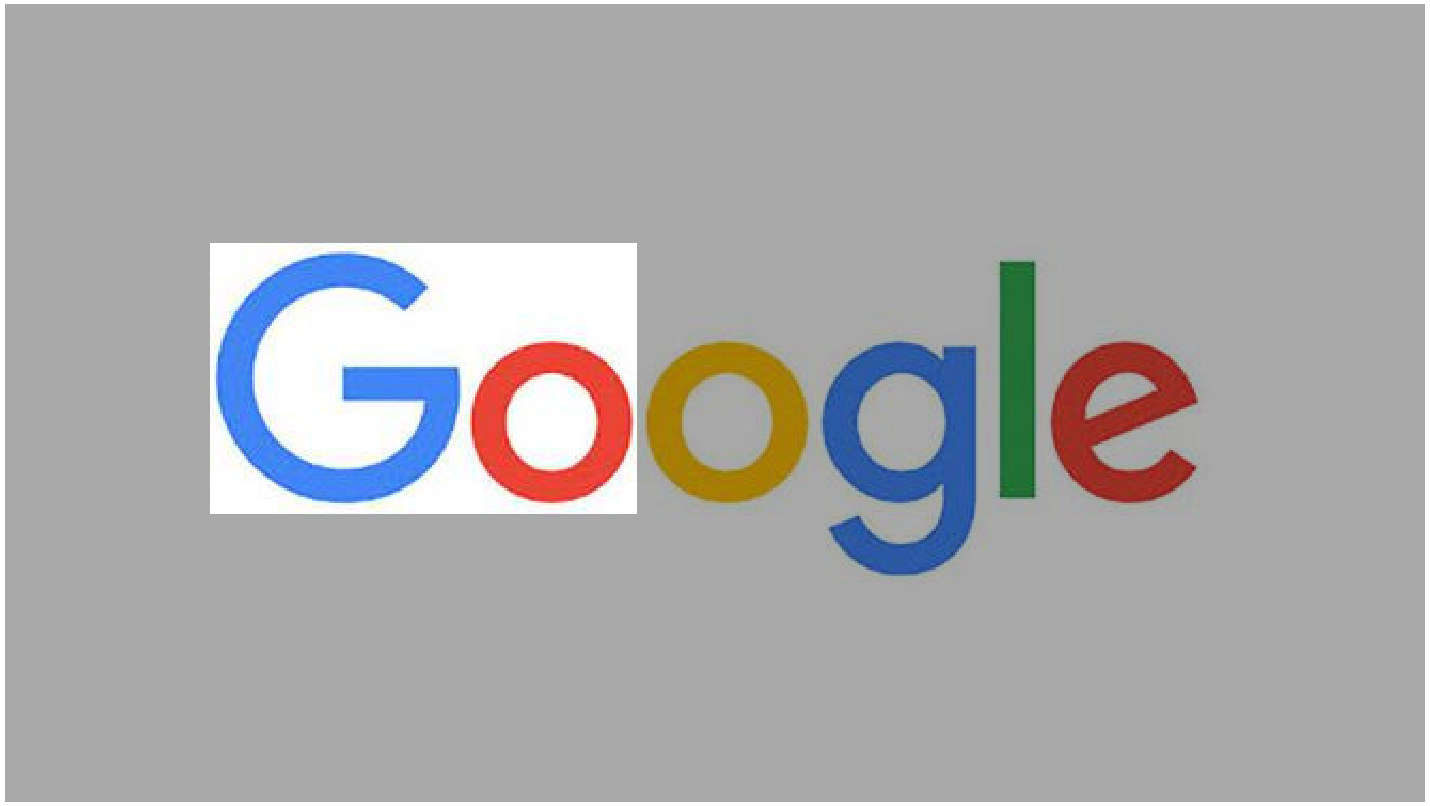
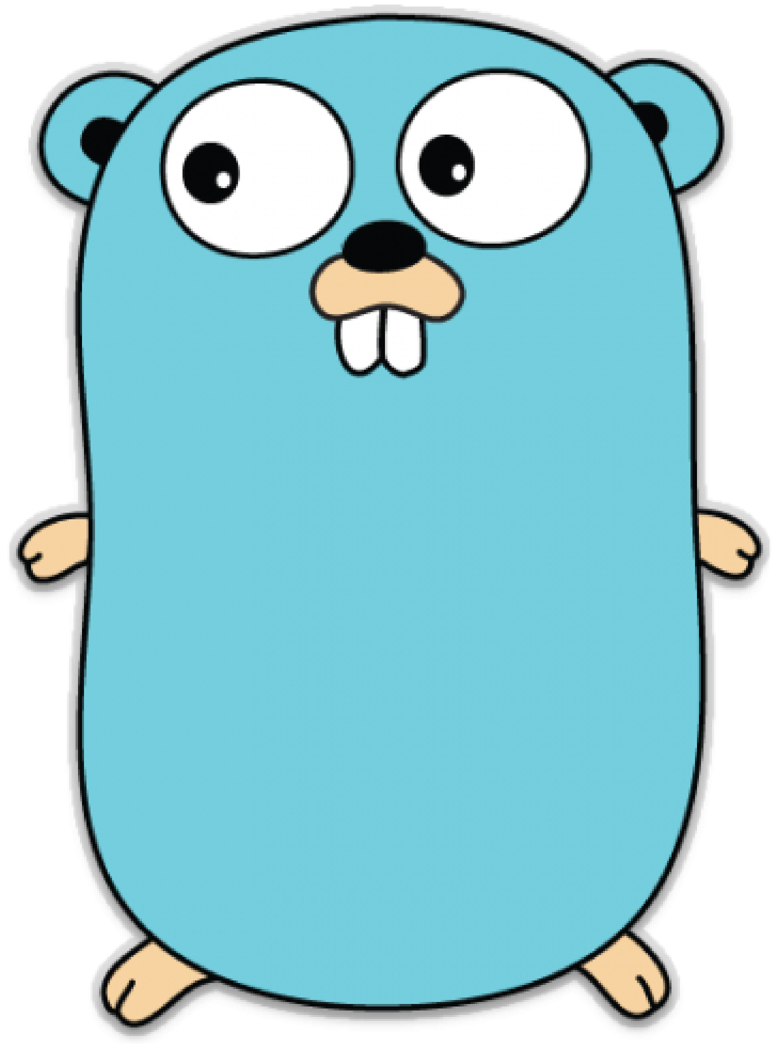
To Go or not to Go

Joseph Buchma
Gopher

Google



Gopher

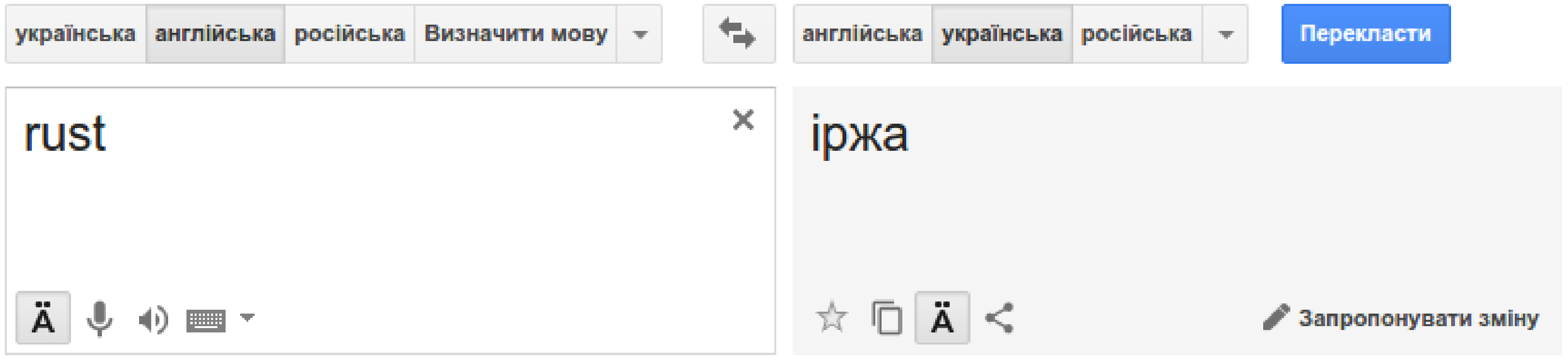


The "Go" game

The motion

"To Go or not to Go?"

To Rust or not to Rust



rust

іржа

Introducing Go

- Compiles to machide code
- Strongly typed
- Concurrency support at the language level
- Garbage collection
- Testing, profiling out of box
- go fmt
- Quick compilation

Hello World

```
package main

import "fmt"

func main() {
    fmt.Println("Hello Wold!")
}
```

Run

Quick overview

GOPATH

The GOPATH environment variable lists places to look for Go code.

What's inside:

- **bin** - compiled binaries goes here
- **pkg** - package objects
- **src** - source code files (third party & your **packages**)

---

go get github.com/stretchr/testify

Import

```go
package mypackage

import (
  "github.com/stretchr/testify"
)
```

Using imported packages

```
In Golang               In python                     Local name of Sin

import   "lib/math"     import lib.math               math.Sin
import m "lib/math"     import lib.math as m          m.Sin
import . "lib/math"     from lib.math import *        Sin
```

Slices and maps

```go
var strSlice []string // unallocated strings slice, eql `nil`.
var xmap map[string]int // == nil
```

Basic usage:

```go
strSlice = make([]string, 5)
xmap = make(map[string]int)

strSlice[0] = 234
strSlice = append(strSlice, 2,4,5)

xmap["foo"] = 123

for key, value := range xmap {
  /* ... */
}
```

Features

Multiple returns (possibly named)

```
func fooBarBaz(n int) (string, bool) {
    var fbb string
    var ok = true

    switch {
    case n%15 == 0:
        fbb = "baz"
    case n%5 == 0:
        fbb = "bar"
    case n%3 == 0:
        fbb = "foo"
    default:
        ok = false
    }
    return fbb, ok
}
```

FooBarBaz test-drive

```go
func fooBarBazDecorated(n int) string {
    if x, ok := fooBarBaz(n); ok {
        return fmt.Sprintf("%s (%d)", x, n)
    }
    return fmt.Sprintf("%d", n)
}

func checkFooBarBaz(n int) {
    for i := 1; i <= n; i++ {
        fmt.Println(fooBarBazDecorated(i))
    }
}
```

Main:

```go
    //checkFooBarBaz(100)
```

Run

Functions are first-class objects

```go
var funcHandler func(int, int)bool

var exit = func(int status) {
  os.Exit(status)
}
```

Closures

```
func checkFBBFunc(n int) func() {
    return func() {
        checkFooBarBaz(n)
    }
}
```

Main:

```
    //fbb30 := checkFBBFunc(30)
    //fbb30()
```

Run

Defer

```go
func somefunc() {
  f, err := os.Open("/path/to/file")
  if err != nil {
    return
  }
  defer f.Close()
  //....
}
```

Custom types & methods

```go
type age uint8

func (a age) String() string {
    return fmt.Sprintf("%d years old", a)
}

type Gopher struct {
    Name string `json:"name"`
    Age  age    `json:"age"`
}

func (g Gopher) Whoami() string {
    return fmt.Sprintf("Hi, my name is %s and I'm %s", g.Name, g.Age)
}

func (g *Gopher) Rename(name string) {
    g.Name = name
}
```

Types & methods showcase

```go
func CustomTypesShowcase() {
    gopher := &Gopher{Name: "Walter White", Age: 50}
    fmt.Println(gopher.Whoami())
    gopher.Rename("Heizenberg")
    fmt.Println(gopher.Whoami())
    fmt.Println(gopher.Age)
}
```

Main:

```go
    //CustomTypesShowcase()
```

Run

Structs embedding

```go
type MyLog struct {
    *log.Logger
    debug bool
}

func (ml MyLog) Debug(msg string) {
    if ml.debug {
        ml.Logger.Print(fmt.Sprintf("[DEBUG] %s", msg))
    }
}

func UseMyLog() {
    logger := MyLog{log.New(os.Stdout, "|logger| ", log.Ltime), true}

    logger.Print("Proxied method of log.Logger")
    logger.Debug("LooooooooooooooL")
}
```

Main:

```go
    //UseMyLog()
```

Run

Interfaces

Specify the behavior of an object

if something can do *this*, then it can be used *here*.
Satisfy interface by defining all methods it requires with exact signatures.

```go
// built in error interface
type error interface {
  Error() string
}

// io.Reader
type Reader interface {
  Read(p []byte) (n int, err error) // reads up to len(p) bytes into p
}
// io.Writer
type Writer interface {
  Write(p []byte) (n int, err error) // writes len(p) bytes from p to the underlying data stream
}
// io.ReadWriter
// Embedding                                      // same as
type ReadWriter interface {                       type ReadWriter interface {
  Reader                                            Read(p []byte) (n int, err error)
  Writer                                            Write(p []byte) (n int, err error)
}                                                 }
```

Example: log.New

log.New's first parameter accepts anything that allows to write into it (anything that implements io.Writer interface)

```
func New(out io.Writer, prefix string, flag int) *Logger
```

Type switch

Type switch

```go
type BusyError struct {
    msg string
}

func (e BusyError) Error() string {
    return e.msg
}

type JessieFailError struct {
    msg    string
    repeat int
}

func (e JessieFailError) Error() string {
    return fmt.Sprintf("Jessie error: %s\n", strings.Repeat(e.msg+" ", e.repeat))
}

func (g *Heizenberg) MakeMeth() error {
    switch rand.Intn(3) {
    case 0:
        return BusyError{"I'm busy"}
    case 1:
        return JessieFailError{"B**ch!", rand.Intn(5) + 1}
    }
    return nil
}
```

The type switch itself

```go
func TypeSwitchExample() {
    H := Heizenberg{}
    for err := H.MakeMeth(); err != nil; err = H.MakeMeth() {
        fmt.Println(err)

        switch err := err.(type) {
        case BusyError:
            fmt.Println("Waiting...")
            time.Sleep(500 * time.Millisecond)
        case JessieFailError:
            fmt.Printf("Teaching Jessie %d times\n", err.repeat)
            time.Sleep(time.Duration(err.repeat) * 100 * time.Millisecond)
        }
        fmt.Println("Trying again...")
    }

    fmt.Println("METH IS DONE!!!")
}
```

Main:

```go
    //TypeSwitchExample()
```

Run

Interface names

By convention, one-method interfaces are named by the method name plus an -er suffix or similar modification
to construct an agent noun: Reader, Writer, Formatter, CloseNotifier etc.

Read, Write, Close, Flush, String and so on have canonical signatures and meanings.
To avoid confusion, don't give your method one of those names unless it has the same signature and meaning.

The bigger the interface, the weaker the abstraction.

Empty interface

```go
func EmptyInterfacesDemo() {
    var foo interface{}
    foo = 3
    foo = "lol"
    foo = Gopher{"John Snow", 30}

    gop, ok := foo.(Gopher)
    fmt.Println(ok)
    fmt.Println(gop.Whoami())

    _, ok = foo.(string)
    fmt.Println(ok)
}
```

Main:

```go
    //EmptyInterfacesDemo()
```

Run

interface{} says nothing

Concurrency

...consists of

- **go** statement (goroutines)
- **channels**
- **select** statement
- **sync** package

Hello Concurrent World!

```go
package main

import "fmt"

func main() {
    done := make(chan struct{})
    go func() {
        fmt.Println("Hello Concurrent World!")
        close(done)
    }()
    <-done
}
```

Run

FooBarBaz HTTP Server

```go
func StartFooBarBazSrv() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        log.Printf("Request %s", r.URL)
        w.Header().Set("Connection", "close")
        i, err := strconv.Atoi(html.EscapeString(r.URL.Path[1:]))
        if err != nil {
            fmt.Fprintf(w, "Invalid number")
        } else {
            fmt.Fprintf(w, fooBarBazDecorated(i))
        }
    })
    log.Print("Serving at localhost:8080")
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

Main:

```go
    //StartFooBarBazSrv()
```

Run

FooBarBaz HTTP Client

```
func GetFBB(n int) (body string, err error) {
    r, err := http.Get(fmt.Sprintf("http://localhost:8080/%d", n))
    if err != nil {
        return
    }
    defer r.Body.Close()
    bbody, err := ioutil.ReadAll(r.Body)
    if err != nil {
        return
    }
    return string(bbody), err
}
```

FooBarBaz HTTP Client

```go
func FBBHTTP(n int) {
    for i := 1; i <= n; i++ {
        if r, e := GetFBB(i); e != nil {
            fmt.Println("Error: %s", e)
        } else {
            fmt.Println(r)
        }
    }
}
```

Main:

```go
    //FBBHTTP(30)
```

Run

FooBarBaz HTTP Concurrent Client

```
func FBBHTTPConcurrent(n int) {
    var wg sync.WaitGroup
    wg.Add(n)
    for i := 1; i <= n; i++ {
        go func(n int) {
            defer wg.Done()
            if r, e := GetFBB(n); e != nil {
                fmt.Println("Error: %s", e)
            } else {
                fmt.Println(r)
            }
        }(i)
    }
    wg.Wait()
}
```

Main:

```
    //FBBHTTPConcurrent(30)
```

Run

FooBarBaz HTTP Concurrent Client Ordered

```
func FBBHTTPConcurrentSorted(n int) {
    returns := make([]chan string, n)
    for i := 1; i <= n; i++ {
        ret := make(chan string)
        returns[i-1] = ret
        go func(n int) {
            if r, e := GetFBB(n); e != nil {
                fmt.Println("Error: %s", e)
                close(ret)
            } else {
                ret <- r
            }
        }(i)
    }
    for _, r := range returns {
        fmt.Println(<-r)
    }
}
```

Main:

```
    //FBBHTTPConcurrentSorted(100)
```

Run

FooBarBaz HTTP Concurrent Client Fan-in Fan-out (1)

```go
func NumGen(n int) <-chan int {
    ch := make(chan int)
    go func() {
        for i := 1; i <= n; i++ {
            ch <- i
        }
        close(ch)
    }()
    return ch
}
```

FooBarBaz HTTP Concurrent Client Fan-in Fan-out (2)

```go
func FBBHTTPConcurrentFanInFanOut(fanOut <-chan int) {
    fanIn := make(chan string)
    for i := 0; i < 100; i++ {
        go func() {
            for n := range fanOut {
                if r, e := GetFBB(n); e != nil {
                    fmt.Println("Error: %s", e)
                } else {
                    fanIn <- r
                }
            }
        }()
    }
    for resp := range fanIn {
        fmt.Println(resp)
    }
}
```

Main:

```go
    //FBBHTTPConcurrentFanInFanOut(NumGen(100000))
```

Run

Explicit cancellation and select statement

```go
func NumGenCancel(n int, done <-chan struct{}) <-chan int {
    ch := make(chan int)
    go func() {
        for i := 1; i <= n; i++ {
            select {
            case ch <- i:
            case <-done:
                return
            }
        }
        close(ch)
    }()
    return ch
}
```

```go
func FBBHTTPConcurrentFanInFanOutCancel(n int) {
    done := make(chan struct{})
    defer close(done)
    fanOut := NumGenCancel(n, done)
    fanIn := make(chan string)
    for i := 0; i < 100; i++ {
        go func() {
            for n := range fanOut {
                if r, e := GetFBB(n); e != nil {
                    fmt.Println("Error: %s", e)
                    return
                } else {
                    fanIn <- r
                }
            }
        }()
    }
    for resp := range fanIn {
        fmt.Println(resp)
    }
}
```

Main:

```
    //FBBHTTPConcurrentFanInFanOutCancel(1000000)
```

Run

Thank you

Joseph Buchma
Gopher

@josephbuchma (http://twitter.com/josephbuchma)