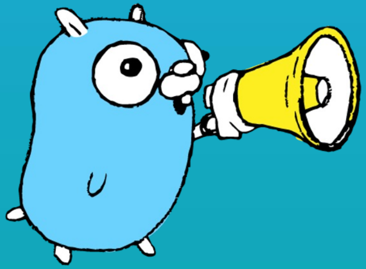
#DevDaysHyd, JUL 29 2023

THINKING IN IDIOMATIC GO

Shubham Sharma

Microsoft

@shubham1172



Today's session

Introduction

Looking at some Go code

Go Proverbs

Tooling

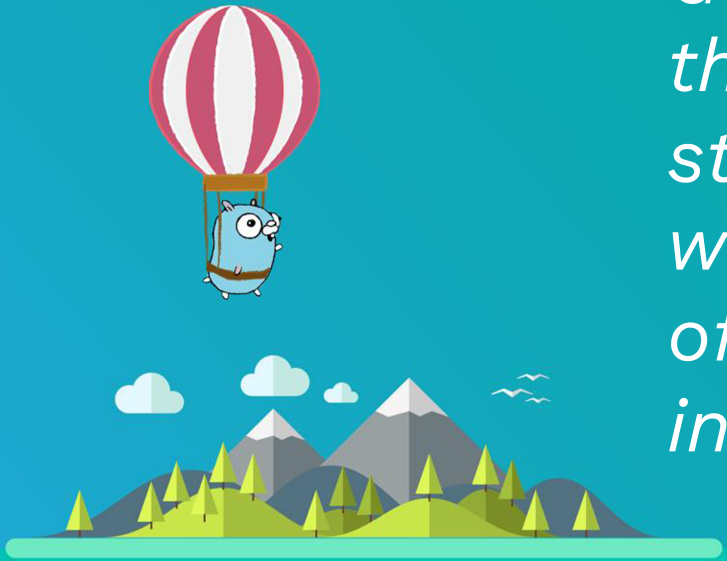
Resources

A BRIEF INTRODUCTION

About Go

Go’s Origins

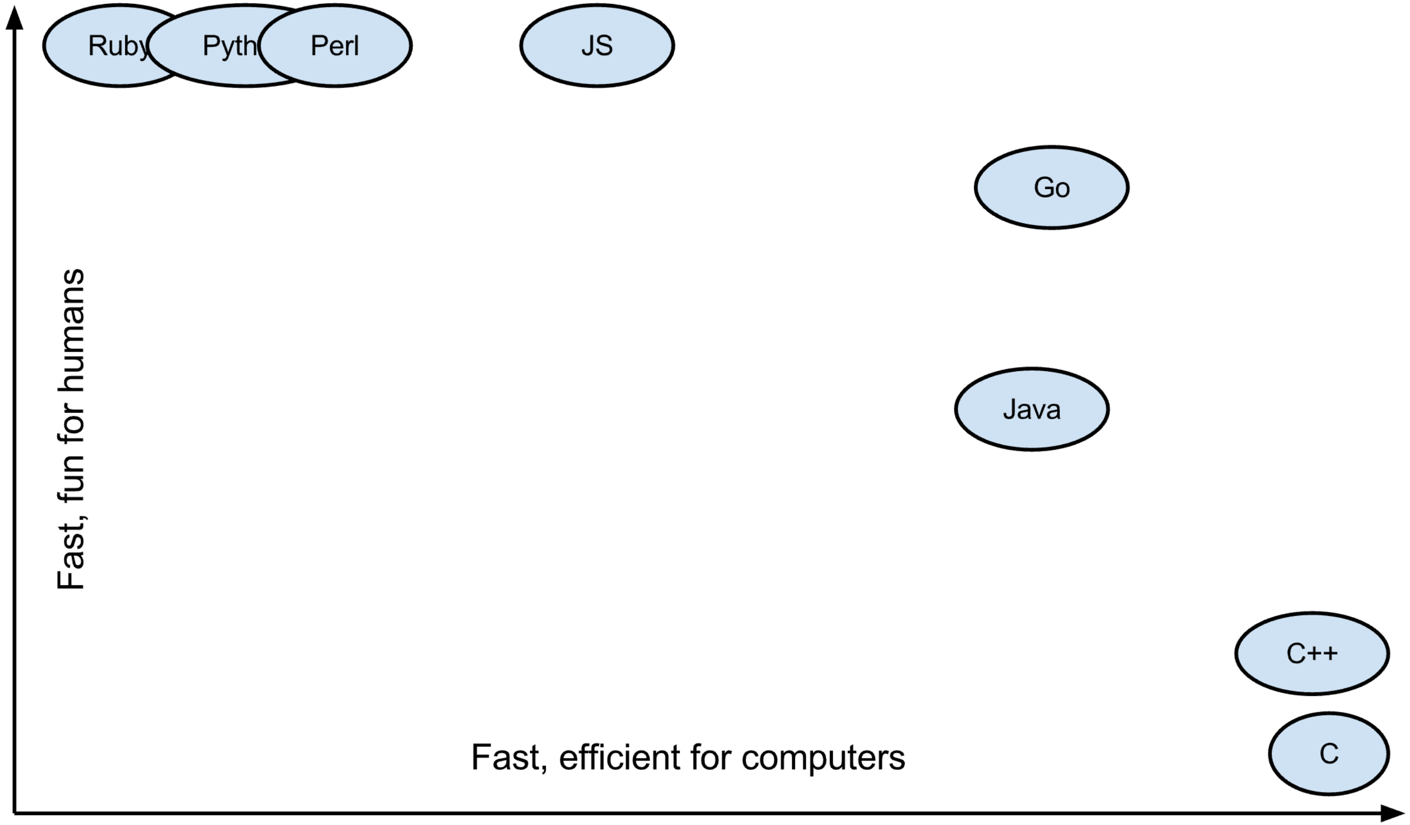
Go is a **statically typed**, **compiled** programming language from the authors of Unix, B, and V8 JavaScript engine.



“Go is an attempt to combine the safety and performance of statically typed languages with the convenience and fun of dynamically typed interpretative languages.”

ROB PIKE

Go is fun and fast



From Brad Fitzpatrick's talk at GoCon Tokyo, 2014

Go’s Values

- Thoughtful
- Simple
- Efficient
- Reliable
- Productive
- Friendly

SOME EXAMPLES

Go code

FUNCTIONS AND METHODS

Not everything requires a struct

```
func add(a, b int) int {
    return a + b
}

func sub(a, b int) int {
    return a - b
}
```

Simple functions wherever possible.

FUNCTIONS AND METHODS

But some do!

```go
type DB struct {
	mu   sync.RWMutex
	data map[string]string
}

func (db *DB) read(key string) (string, error) {
	db.mu.RLock()
	defer db.mu.RUnlock()

	val, ok := db.data[key]
	if !ok {
		return "", fmt.Errorf("Key %s not found!", key)
	}

	return val, nil
}
```

Use methods when

1. Naturally tied to a struct
2. Accessing state
3. Implementing interfaces

GO HAPPY PATH

Easier to read and differentiate between core logic and error handling

```go
func divide(a, b int) (int, error) {
    if b == 0 {
        return -1, fmt.Errorf("Cannot divide by zero!")
    } else {
        return a / b, nil
    }
}
```

```go
func divide(a, b int) (int, error) {
    if b == 0 {
        return -1, fmt.Errorf("Cannot divide by zero!")
    }

    return a / b, nil
}
```

Happy path
is left-aligned.

GLOBAL VARIABLES ARE NOT ALWAYS BAD

Example from standard library “net/http”

```go
var (
	ErrBodyNotAllowed = errors.New("http: request method or response status code does not allow body")
	ErrHijacked = errors.New("http: connection has been hijacked")
)

func (w *response) write(lenData int, dataB []byte, dataS string) (n int, err error) {
	// ...
	if !w.bodyAllowed() {
		return 0, ErrBodyNotAllowed
	}
	// ...
}
```

Do not mutate global variables.

GLOBAL VARIABLES ARE NOT ALWAYS BAD

Example from Cobra – a popular CLI framework

```go
var rootCmd = &cobra.Command{
  Use:   "hugo",
  Short: "Hugo is a very fast static site generator",
  Long: `A Fast and Flexible Static Site Generator built with
                love by spf13 and friends in Go.
                Complete documentation is available at http://hugo.spf13.com`,
  Run: func(cmd *cobra.Command, args []string) {
    // Do Stuff Here
  },
}

func Execute() {
  if err := rootCmd.Execute(); err != nil {
    fmt.Println(err)
    os.Exit(1)
  }
}
```

Go Proverbs!

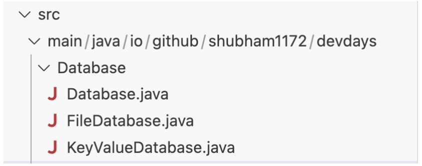
“The bigger the interface, the weaker the abstraction.”

OOP - CLASS IMPLEMENTS INTERFACE

Create interfaces first, implementation second

Interface and implementation live together.

```java
interface Database {
  String read(String key);
  void write(String key, String value);
}
```



```
∨ src
  ∨ main / java / io / github / shubham1172 / devdays
    ∨ Database
      J Database.java
      J FileDatabase.java
      J KeyValueDatabase.java
```

```java
import java.util.HashMap;

class KeyValueDatabase implements Database {
  private HashMap<String, String> keyValueStore;

  KeyValueDatabase() {
    keyValueStore = new HashMap<String, String>();
  }

  public String read(String key) {
    return keyValueStore.get(key);
  }

  public void write(String key, String value) {
    keyValueStore.put(key, value);
  }
}
```

```java
package io.github.shubham1172.devdays;

import io.github.shubham1172.devdays.Database.Database;
import io.github.shubham1172.devdays.Database.KeyValueDatabase;

public final class App {
    private static void printValueFromDatabase(Database db, String key) {
        System.out.println("Value for key " + key + " is " + db.read(key));
    }

    public static void main(String[] args) {
        KeyValueDatabase db = new KeyValueDatabase();
        db.write("key1", "value1");

        printValueFromDatabase(db, "key1");
    }
}
```

GO - TYPE SATISFIES INTERFACE

Write types to meet expectations

Interfaces are defined where it is used.

```go
package database

import (
    "errors"
    "sync"
)

var ErrNotFound = errors.New("key not found")

type KeyValueDB struct {
    data sync.Map
}

func (db *KeyValueDB) Read(key string) (string, error) {
    value, ok := db.data.Load(key)
    if !ok {
        return "", ErrNotFound
    }
    return value.(string), nil
}

func (db *KeyValueDB) Write(key, value string) error {
    db.data.Store(key, value)
    return nil
}
```

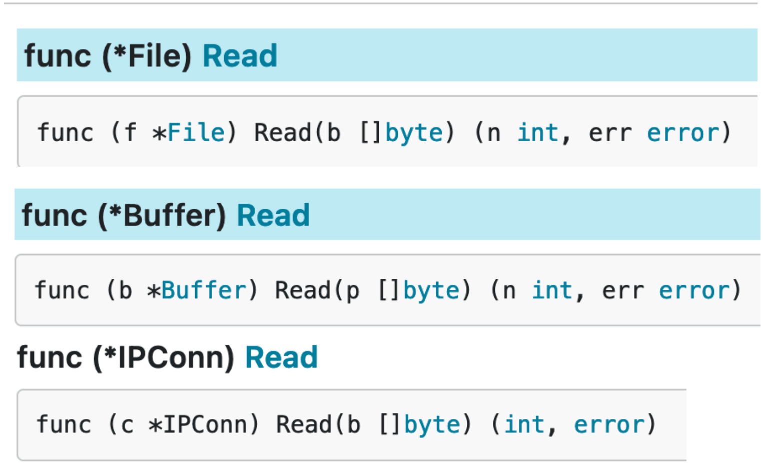
```go
type Reader interface {
    Read(string) (string, error)
}

func tryPrintValue(r Reader, key string) {
    value, err := r.Read(key)
    if err != nil {
        fmt.Printf("Error: %s\n", err)
    }

    fmt.Printf("Value: %s\n", value)
}

func main() {
    kvdb := &database.KeyValueDB{}
    tryPrintValue(kvdb, "hello")
}
```

GO – A LOOK AT IO.READER

A small interface with a strong abstraction!

```go
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

Use well-defined abstractions in your library!



func (*File) Read

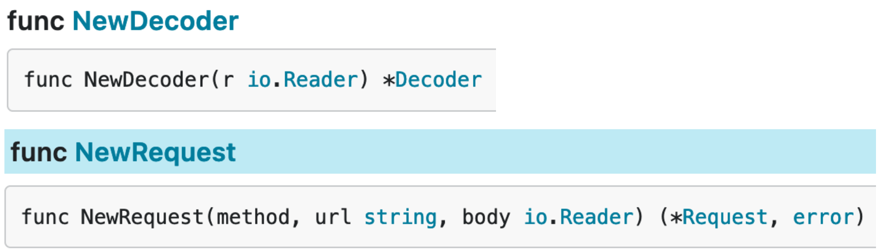
```go
func (f *File) Read(b []byte) (n int, err error)
```

func (*Buffer) Read

```go
func (b *Buffer) Read(p []byte) (n int, err error)
```

func (*IPConn) Read

```go
func (c *IPConn) Read(b []byte) (int, error)
```



func NewDecoder

```go
func NewDecoder(r io.Reader) *Decoder
```

func NewRequest

```go
func NewRequest(method, url string, body io.Reader) (*Request, error)
```

```go
// limited to strings
func Reverse(s string) (string, error)

// can reverse anything that satisfies io.Reader
func Reverse(r io.Reader) (io.Reader, error)
```

https://pkg.go.dev/io#Reader

“A little copying is better than a little dependency”

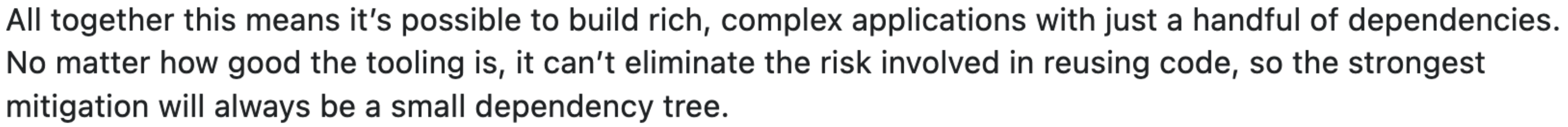
“Clear is better than clever”

BE CAREFUL WITH DEPENDENCIES

A little copying's not a crime
It can save you from dependency's grime
Just be sure to give credit
Where credit is due
And avoid any potential legal time.

https://appliedgo.net/limericks/

Dependency hygiene trumps code reuse.


All together this means it's possible to build rich, complex applications with just a handful of dependencies. No matter how good the tooling is, it can't eliminate the risk involved in reusing code, so the strongest mitigation will always be a small dependency tree.

https://go.dev/blog/supply-chain

CLEAR IS BETTER THAN CLEVER

Less is exponentially more

"Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?”
~ Brian Kernighan

“Errors are values”

“Don't just check errors, handle them gracefully”

ERRORS IN GO

Error indicates an abnormal state.

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


func Sqrt(f float64) (float64, error) {
    if f < 0 {
        // use built-in error type to create a new error
        return 0, errors.New("math: square root of negative number")
    }
    // implementation
}


// Or implement your own error!
type NegativeSqrtError float64

func (f NegativeSqrtError) Error() string {
    return fmt.Sprintf("math: square root of negative number %g", float64(f))
}
```

Errors are treated just like values.

```go
f, err := Sqrt(-1)
if err != nil {
    fmt.Println(err)
}
```

Go tooling

GO TOOLS

- go mod, go get
- go build, go install
- go test
- gofmt, goimports
- godoc
- trace
- pprof

“Gofmt's style is no one's favorite, yet gofmt is everyone's favorite.”

NEXT STEPS

What now?

LEARNING MATERIAL

Gopherfest 2015 | Go Proverbs with Rob Pike

Effective Go | Go documentation

When in Go, do as Gophers do | GoCon 2014

CodeReviewComments Go

THANK YOU