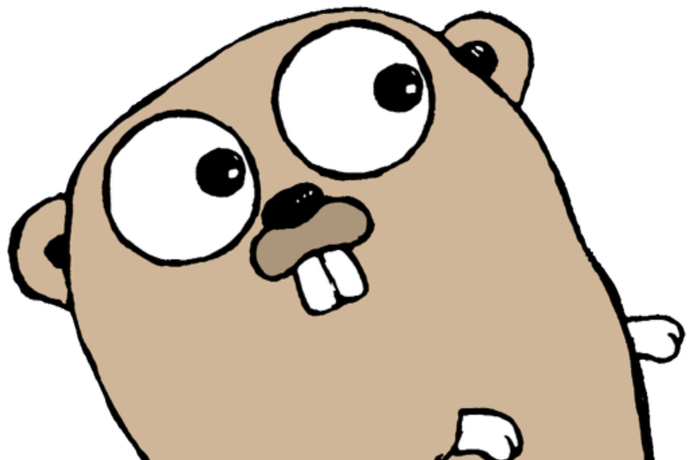
# Go: A Pragmatic Language

Anthony Eden - @aeden - DNSimple



prag·mat·ic - “dealing with things sensibly and realistically in a way that is based on practical rather than theoretical considerations.”

*“Go's purpose is not to do research into programming language design; it is to improve the working environment for its designers and their coworkers.”*

# What is Go?

# Systems Language

# Strongly Typed

# Garbage Collected

# Explicit support for Concurrent Programming

*“Go is more about software engineering than programming language research. Or to rephrase, it is about language design in the service of software engineering.”*

# Learning by Example

# HTTP Server

```go
package main

import



)

func
        fmt.Fprintf(w,
}

func
        http.HandleFunc(

        log.Fatal(http.ListenAndServe(
}
```

```go
package main

import (
	"fmt"
	"log"
	"net/http"
)

func rootHandler(w http.ResponseWriter, r *http.Request) {
	fmt.Fprintf(w, "All systems go\n")
}

func main() {
	http.HandleFunc("/", rootHandler)

	log.Fatal(http.ListenAndServe(":8080", nil))
}
```

# HTTP Client

```
package main

import


)

func
        resp, err

                log.Printf(
        }
                log.Printf(
        }
}
```

```go
package main

import (
	"log"
	"net/http"
)

func main() {
	resp, err := http.Get("http://localhost:8080/")
	if err != nil {
		log.Printf("Error: %s", err)
	} else {
		log.Printf("%v", resp)
	}
}
```

# Initialization: Constants, Variables and init()

```go
const DEFAULT_HTTP_BIND_PORT = "8080"

var (
        httpBindAddress = os.Getenv("HTTP_BIND_ADDRESS")
        httpBindPort    = os.Getenv("HTTP_BIND_PORT")
)

func init() {
        if httpBindPort == "" {
                httpBindPort = DEFAULT_HTTP_BIND_PORT
        }
}
```

```
package

import




)

const DEFAULT_HTTP_BIND_PORT = "8080"

var (
        httpBindAddress = os.Getenv("HTTP_BIND_ADDRESS")
        httpBindPort    = os.Getenv("HTTP_BIND_PORT")
)

func init() {
        if httpBindPort == "" {
                httpBindPort = DEFAULT_HTTP_BIND_PORT
        }
}

func rootHandler(w http.ResponseWriter, r
        fmt.Fprintf(w,
}

func main() {
        http.HandleFunc(

        hostAndPort
        log.Fatal(http.ListenAndServe(hostAndPort,
}
```

# Types

# Slices

```go
a := []int{1, 2, 3, 4, 5} // [1 2 3 4 5]
```

# Looping

```go
i := 0
for {
        fmt.Printf("%v: %v\n", i, a[i])
        i += 1
        if i >= len(a) {
                break
        }
}
```

```go
for index, value := range a {
        fmt.Printf("%v: %v\n", index, value)
}
```

# Functions

```go
func rootHandler(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "All systems go\n")
}
```

```go
f := func(x int, y int) int {
        return x + y
}
fmt.Printf("Function result: %v\n", f(10, 22))
```

```go
func main() {
        a := []int{1, 2, 3, 4, 5}









}
```

“Go is an attempt to combine the ease of programming of an interpreted, dynamically typed language with the efficiency and safety of a statically typed, compiled language.”

```
go build server.go
```

```
go run server.go
```

# Dependency Management

```go
package gapl

type Person struct {
        Name string
        Age  int
}
```

```go
import (
        "encoding/json"
        "fmt"
        gapl "github.com/aeden/go-a-pragmatic-language-lib"
        "log"
        "net"
        "net/http"
        "os"
)
```

```
import

        gapl "github.com/aeden/go-a-pragmatic-language-lib"

)
```

```go
func personHandler(w http.ResponseWriter, r *http.Request) {
        w.Header().Set("Content-type", "application/json")
        person := gapl.Person{Name: "John Smith", Age: 42}
        encoder := json.NewEncoder(w)
        encoder.Encode(person)
}
```

```go
func personHandler(w http.ResponseWriter, r *http.Request) {
        w.Header().Set(
        person
        encoder
        encoder.Encode(person)
}
```

```go
package

import

        gapl "github.com/aeden/go-a-pragmatic-language-lib"



)

const DEFAULT_HTTP_BIND_PORT =

var (
        httpBindAddress = os.Getenv(
        httpBindPort    = os.Getenv(
)

func init() {



}

func rootHandler(w http.ResponseWriter, r
        fmt.Fprintf(w,
}

func personHandler(w http.ResponseWriter, r *http.Request) {
        w.Header().Set("Content-type", "application/json")
        person := gapl.Person{Name: "John Smith", Age: 42}
        encoder := json.NewEncoder(w)
        encoder.Encode(person)
}

func main() {
        http.HandleFunc(
        http.HandleFunc("/person", personHandler)

        hostAndPort
        log.Fatal(http.ListenAndServe(hostAndPort,
}
```

# Concurrency

# Communicating Sequential Processes

# Message Passing Via Channels

```go
package main

import (
        "fmt"
        "math/rand"
        "time"
)
```

```go
func addRand(c chan int) {
        for {
                time.Sleep(1 * time.Second)
                c <- rand.Intn(100)
        }
}

func subRand(c chan int) {
        for {
                time.Sleep(2 * time.Second)
                c <- -rand.Intn(100)
        }
}
```

```go
func printSum(c chan int) {
        sum := 0
        val := 0

        for {
                val = <-c
                fmt.Printf("%v + %v = %v\n", sum, val, sum+val)
                sum = sum + val
        }
}
```

```go
func main() {

}
```

```go
package main

import (
    "fmt"
    "math/rand"
    "time"
)

func addRand(c chan int) {
    for {
        time.Sleep(1 * time.Second)
        c <- rand.Intn(100)
    }
}

func subRand(c chan int) {
    for {
        time.Sleep(2 * time.Second)
        c <- -rand.Intn(100)
    }
}

func printSum(c chan int) {
    sum := 0
    val := 0

    for {
        val = <-c
        fmt.Printf("%v + %v = %v\n", sum, val, sum+val)
        sum = sum + val
    }
}

func main() {
    rand.Seed(time.Now().Unix())

    quitChan := make(chan int)

    c := make(chan int)

    go printSum(c)
    go addRand(c)
    go subRand(c)

    <-quitChan
}
```

# Interfaces

# Test

```
package gapl
```

# Implementation

```
package gapl
```

# Where Is Go Useful?

# Network Services

# System Monitoring Agents

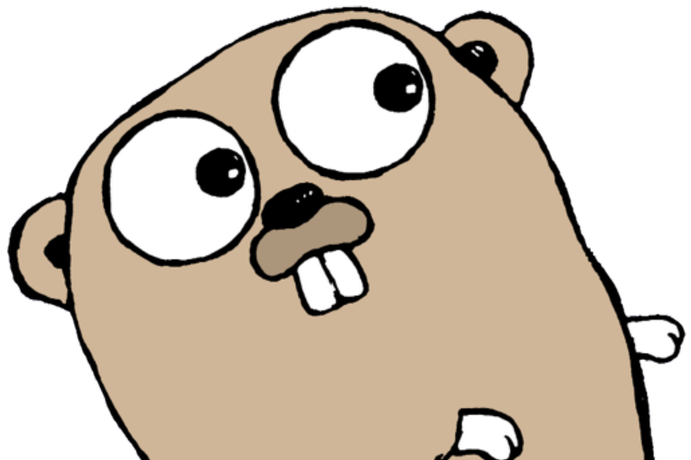
# Command-line Utilities

# Data Processing

# Let’s See it Work

“Go is an attempt to combine the ease of programming of an interpreted, dynamically typed language with the efficiency and safety of a statically typed, compiled language.”

- What systems do I already control that might benefit from Go's design?
- What can I learn from Go to take to my preferred programming environment?
- What is missing from Go that I absolutely cannot live without, and why?

# Go: A Pragmatic Language

Anthony Eden - @aeden - DNSimple