Building Go CLI APP

Generating Randomized Data

...by Abhisek Pattnaik
Fullstack Software Engineer
about.me/abhisekp

Prerequisites and General Info

- ➔ Knowledge about Go
- ➔ VSCode or Goland IDE
- ➔ JSON Basics
- ➔ XML Basics

- ➔ All relevant links in the slide are clickable
- ➔ All code images are clickable and point to the repository with appropriate commit
- ➔ Slide and the CLI app Project are published in Speakerdeck and Github respectively
- ➔ Relevant code commits are tagged as per the slide#

TABLE OF CONTENTS

01

Introduction to CLI APP

Why build CLI apps in Go?

- ➔ Build Reliable, efficient, and scalable CLI Apps
- ➔ Build fast and with ease
- ➔ Rich libraries for tasks like manipulating text, files, and data
- ➔ Built-in and third party modules for parsing CLI arguments with ease

What is CLI?

- ➔ Command-Line Interface (CLI)
- ➔ Launch programs and automate with commands
- ➔ Easy to write and understand
- ➔ Less complex than GUI (Graphics User Interface)

Overview of the “flag” module

- ➔ Implements command line flag parsing
- ➔ Define flags with various datatypes
- ➔ Specify default values and usage briefing
- ➔ Parse custom data structures

02

Getting Started with the “flag” module

Syntax for passing flags in CLI app

Syntax

```
-flag

// double dashes are also permitted
--flag

-flag=x

// non-boolean flags only
-flag x
```

Passing flags in CLI

```
▶ ./bin/hello-world
  -n string
        The name to say hello to.
  -name string
        The name to say hello to.
(miniconda3-latest)
Personal/go/conf42-golang-2023-building-cli-app
▶ ./bin/hello-world -n "Abhisek"
Hello, Abhisek
```

Parsing user input from CLI args

main.go

```go
package main

import "flag"

func main() {
        name := ""
        flag.StringVar(&name, "name", "World", "The name to say hello to.")
        flag.Parse()
        println("Hello", name)
}
```

03

Parsing Data with “flag” module

Parsing various datatypes

- `flag.StringVar`
- `flag.IntVar`
- `flag.BoolVar`
- `flag.DurationVar`
- `flag.Float64Var`
- `flag.Var`
- `flag.Func`
- …

Aliases for CLI flags

- ➔ Helps in writing short letters for longer flag names
- ➔ E.g. “-n” instead of “--name” or “-r” instead of “--repeat”, etc.

main.go

```go
package main

import (
	"flag"
	"fmt"
	"os"
)

func main() {
	var name string
	flag.StringVar(&name, "name", "", "The name to say hello to.")
	flag.StringVar(&name, "n", "", "The name to say hello to.")

	flag.Parse()

	if name == "" {
		flag.PrintDefaults()
		os.Exit(0)
	}
	fmt.Println("Hello,", name)
}
```

```
▶ ./bin/hello-world
  -n string
        The name to say hello to.
  -name string
        The name to say hello to.
(miniconda3-latest)
Personal/go/conf42-golang-2023-building-cli-app
▶ ./bin/hello-world -n "Abhisek"
Hello, Abhisek
```

Better aliasing with “getopt” library

- ➔ No redundant flags in help text
- ➔ Improve maintainability
- ➔ rsc.io/getopt

main.go

```go
package main

import (
	"flag"
	"fmt"
	"os"

	"rsc.io/getopt"
)

func main() {
	name := ""
	flag.StringVar(&name, "name", "", "The name to say hello to.")
	// flag.StringVar(&name, "n", "", "The name to say hello to.")
	getopt.Aliases(
		"n", "name",
	)

	getopt.Parse()

	if name == "" {
		getopt.PrintDefaults()
		os.Exit(0)
	}
	fmt.Println("Hello,", name)
}
```

```
Personal/go/conf42-golang-2023-building-cli-app
► ./bin/hello-world -n "Abhisek"
Hello, Abhisek
(miniconda3-latest)
Personal/go/conf42-golang-2023-building-cli-app
► ./bin/hello-world
  -n, --name string
        The name to say hello to.
```

Getting Help in CLI app

- ➔ Provides brief overview about the CLI flags
- ➔ Use `` `flag.PrintDefaults()` `` to print the usage

main.go

```go
package main

import (
	"flag"
	"os"

	"rsc.io/getopt"
)

func main() {
	totalArgs := len(os.Args[1:])

	var help bool
	flag.BoolVar(&help, "help", totalArgs == 0, "Show help")

	getopt.Aliases(
		"h", "help",
	)

	getopt.Parse()

	if help {
		getopt.PrintDefaults()
		os.Exit(0)
	}
}
```

```
Personal/go/conf42-golang-2023-building-cli-app
► ./bin/hello-world
  -h, --help
        Show help (default true)
(miniconda3-latest)
Personal/go/conf42-golang-2023-building-cli-app
► ./bin/hello-world -h
  -h, --help
        Show help
```

Naming your CLI APP

- ➔ Use `` `flag.NewFlagSet` `` to create new flagset
- ➔ Specify name of the CLI and error handling type

```
▶ ./bin/hello-world --name "Abhisek"
Hello, Abhisek
(miniconda3-latest)
Personal/go/conf42-golang-2023-building-cli-app

▶ ./bin/hello-world -n
flag provided but not defined: -n
Usage of Abhisek's CLI:
  -name string
        The name to say hello to.
```

main.go

```go
package main

import (
  "flag"
  "os"

  "rsc.io/getopt"
)

func main() {
  totalArgs := len(os.Args[1:])

  fs := getopt.NewFlagSet("Abhisek's CLI", flag.ExitOnError)

  var help bool
  fs.BoolVar(&help, "help", totalArgs == 0, "Show help")

  fs.Aliases(
    "h", "help",
  )

  err := fs.Parse(os.Args[1:])
  if err != nil {
    return
  }

  if help {
    fs.Usage()
    os.Exit(0)
  }
}
```

Parsing as structured data

- Create your datatype struct
- Use the struct as pointer field in a “namedValue” struct
- Implement the following interface

```
type Value interface {
    String() string
    Set(string) error
}
```

- The `String()` method returns the stringified value of the data struct.
- The `Set(string) error` should parse the value and return any error.

Parsing as structured data (Demo)

main.go

```go
package main

import (
	"errors"
	"flag"
	"fmt"
	"strings"
)

type Address struct {
	Street  string
	City    string
	State   string
	Pincode string
	Country string
}

type AddressValue struct {
	*Address
}

func (a AddressValue) String() string {
	return fmt.Sprintf("%s, %s, %s, %s, %s", a.Address.Street, a.Address.City,
a.Address.State, a.Address.Pincode, a.Address.Country)
}
```

main.go

```go
func (a AddressValue) Set(addressStr string) error {
	addressParts := strings.Split(addressStr, ", ")
	if len(addressParts) == 0 || len(addressParts) > 5 {
		return errors.New("Address should be in the format: Street, City, State,
Pincode, Country")
	}

	if len(addressParts) >= 1 {
		a.Address.Street = addressParts[0]
	}

	if len(addressParts) >= 2 {
		a.Address.City = addressParts[1]
	}

	if len(addressParts) >= 3 {
		a.Address.State = addressParts[2]
	}

	if len(addressParts) >= 4 {
		a.Address.Pincode = addressParts[3]
	}

	if len(addressParts) >= 5 {
		a.Address.Country = addressParts[4]
	}

	return nil
}

func main() {
	var address Address
	addressVar := AddressValue{&address}
	flag.Var(&addressVar, "address", "The address of the person in the format: Street,
City, State, Pincode, Country")

	flag.Parse()

	fmt.Println("Address")
	fmt.Printf("%q\n", address)
}
```

Extracting selective data

➔ Define a function to parse the string passed to the flag

main.go

```go
package main

import (
	"errors"
	"flag"
	"fmt"
	"regexp"
	"strconv"
)

var reNum = regexp.MustCompile(`\d+`)

func main() {
	var num int
	flag.Func("num", "number of records to generate", func(s string) error {
		var err error
		num, err = strconv.Atoi(reNum.FindString(s))
		if err != nil {
			return errors.New("string must have number")
		}
		return nil
	})

	flag.Parse()

	fmt.Println("num:", num)
}
```

```
▶ ./bin/hello-world --num a334sfsd
num: 334
(miniconda3-latest)
Personal/go/conf42-golang-2023-building-cli-app
▶ ./bin/hello-world
num: 0
(miniconda3-latest)
Personal/go/conf42-golang-2023-building-cli-app
▶ ./bin/hello-world -h
Usage of ./bin/hello-world:
  -num value
        number of records to generate
▶ ./bin/hello-world --num asdada
invalid value "asdada" for flag -num: string must have number
Usage of ./bin/hello-world:
  -num value
        number of records to generate
```

Input json file (with multiple types) and error

main.go

```
{
  "name": "The National Archives",
  "items": [
    {
      "name": "Benjamin",
      "quantity": 1,
      "percentage": 0.14,
      "aliases": "Benjamin, Ben, Benjy, Benj, Be"
    },
    {
      "name": "John",
      "quantity": 2,
      "percentage": 0.28,
      "active": true,
      "aliases": [
        "Johnny",
        "Johnnie",
        "Jack",
        "Jock"
      ]
    },
    {
      "name": "Gerard",
      "quantity": 3,
      "percentage": 0.42,
      "active": true,
      "aliases": [
        "Gera",
        "Ger",
        "Gerry",
        "Gerrard"
      ]
    }
  ]
}
```

```
▶ ./bin/json-cli -i ./input.json
name: Benjamin
quantity: 1
percentage: 0.14
active: false
aliases: Benjamin, Ben, Benjy, Benj, Be

name: John
quantity: 2
percentage: 0.28
active: true
aliases: Johnny, Johnnie, Jack, Jock
```

```
▶ ./bin/json-cli -i ./inputError.json
invalid value "./inputError.json" for flag -i: invalid character '{' after object key:value pair
Usage of JSON Input CLI:
  -i, --input value
        Input JSON
```

Parsing user input from json file

- ➔ Pass the file input path to read from the file and parse
- ➔ Implement Unmarshaller for the JSON struct
- ➔ Implement `` `flag.Value` `` interface to parse structured data
- ➔ Use `` `flag.Var()` `` to input custom data structure for parsing

Parsing user input from json file (Demo)

main.go

```go
package main

import (
	"encoding/json"
	"flag"
	"fmt"
	"os"
	"path/filepath"
	"regexp"
	"strings"

	"github.com/ImpelsysInc/go-ptr"
	"rsc.io/getopt"
)

func main() {
	var input Input

	fs := getopt.NewFlagSet("JSON Input CLI", flag.ExitOnError)

	fs.Var(&jsonInputFlag{&input}, "input", "Input JSON")
	fs.Aliases("i", "input")

	err := fs.Parse(os.Args[1:])
	if err != nil {
		os.Exit(2)
	}

	fmt.Println()
	for _, item := range input.Items {
		fmt.Println("name:", ptr.StringValue(item.Name))
		fmt.Println("quantity:", ptr.IntValue(item.Quantity))
		fmt.Println("percentage:", ptr.Float32Value(item.Percentage))
		fmt.Println("active:", ptr.BoolValue(item.Active))
		fmt.Println("aliases:", strings.Join(item.Aliases, ", "))
		fmt.Println()
	}
}
```

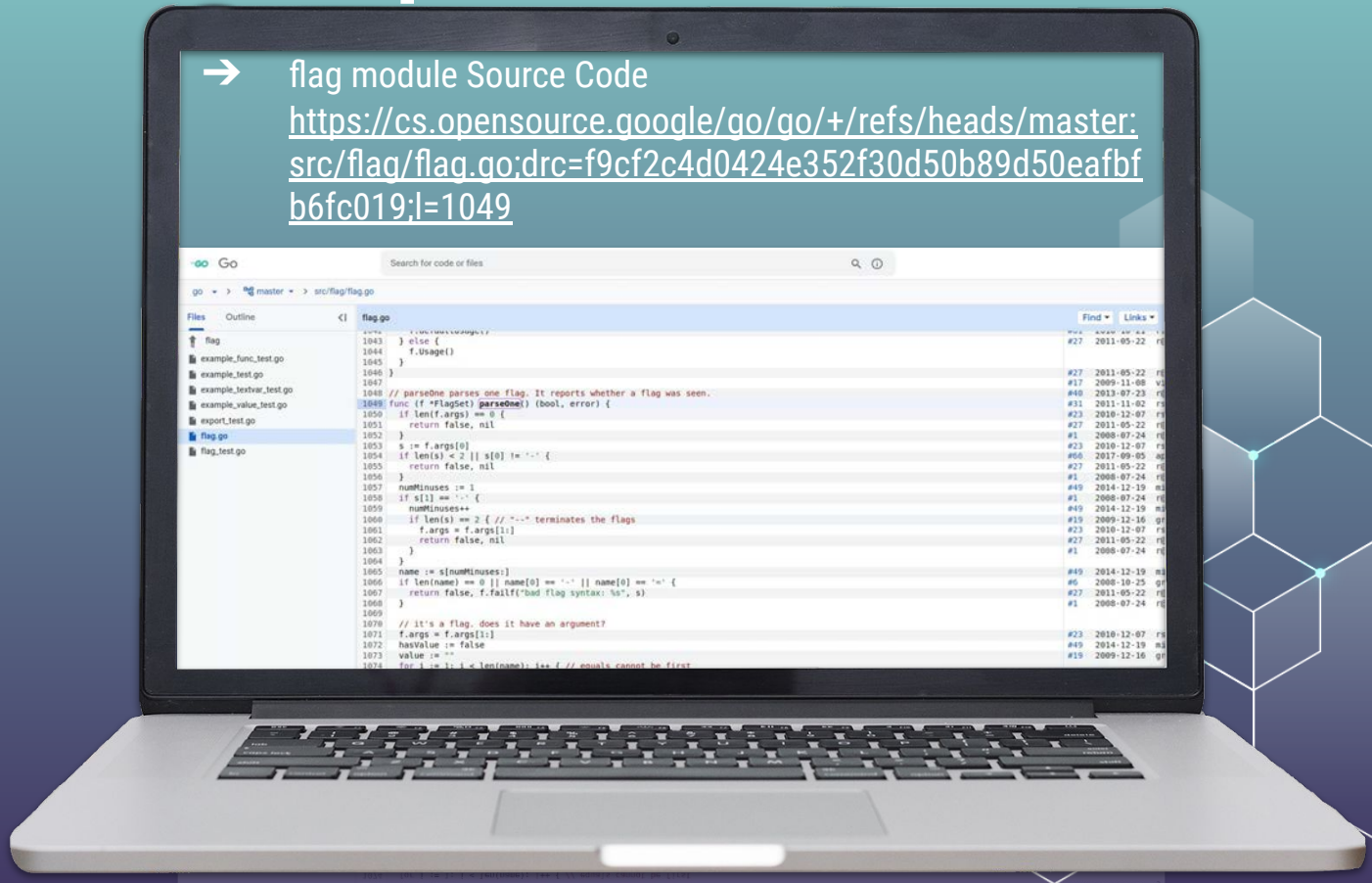
main.go

```go
type Input struct {
	Name  string  `json:"name"`
	Items []*Item `json:"items"`
}

type Item struct {
	Name       *string  `json:"name"`
	Quantity   *int     `json:"quantity"`
	Percentage *float32 `json:"percentage"`
	Active     *bool    `json:"active"`
	Aliases    []string `json:"aliases"`
}

func (i *Item) UnmarshalJSON(data []byte) error {
	type Alias Item
	aux := &struct {
		Aliases any `json:"aliases"`
		*Alias
	}{
		Alias: (*Alias)(i),
	}
	if err := json.Unmarshal(data, &aux); err != nil {
		return err
	}
	switch aliases := aux.Aliases.(type) {
	case string:
		reComma := regexp.MustCompile(`\s*,\s*`)
		i.Aliases = reComma.Split(aliases, -1)
	case []any:
		i.Aliases = make([]string, len(aliases))
		for j, alias := range aliases {
			i.Aliases[j] = fmt.Sprintf("%v", alias)
		}
	}
	return nil
}
```

main.go

```go
type jsonInputFlag struct {
  inputFile *Input
}

func (f *jsonInputFlag) String() string {
  return ""
}

func (f *jsonInputFlag) Set(value string) error {
  abs, err := filepath.Abs(value)
  if err != nil {
    return err
  }

  // Read the JSON file
  file, err := os.ReadFile(abs)
  if err != nil {
    return err
  }

  // Parse the JSON
  err = json.Unmarshal(file, f.inputFile)
  if err != nil {
    return err
  }

  return nil
}
```

flag module implementation (internal)



- flag module Source Code
  https://cs.opensource.google/go/go/+/refs/heads/master:src/flag/flag.go;drc=f9cf2c4d0424e352f30d50b89d50eafbfb6fc019;l=1049

04

Generating Multiple Random Data

Random Number Generation using CLI

- ➔ Input min and max from CLI argument
- ➔ Pass cli input to random number generator

main.go

```go
package main

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

	"rsc.io/getopt"
)

func main() {
	min, max := 0, 1

	flag.IntVar(&min, "min", min, "Minimum value")
	flag.IntVar(&max, "max", max, "Maximum value")

	getopt.Aliases(
		"m", "min",
		"M", "max",
	)

	getopt.Parse()

	// Seed the random number generator with the current time
	rnd := rand.New(rand.NewSource(time.Now().UnixNano()))

	randomVal := min + rnd.Intn(max-min)
	fmt.Println(randomVal)
}
```

```
Personal/go/conf42-golang-2023-building-cli-app
▶ ./bin/rand-cli
0
(miniconda3-latest)
Personal/go/conf42-golang-2023-building-cli-app
▶ ./bin/rand-cli -m 10 -M 100
27
(miniconda3-latest)
Personal/go/conf42-golang-2023-building-cli-app
▶ ./bin/rand-cli -m 10 -M 100
21
(miniconda3-latest)
Personal/go/conf42-golang-2023-building-cli-app
▶ ./bin/rand-cli -m 10 -M 100
58
```

Random Data Generation using faker

➔ Use faker library to generate random data
https://github.com/brianvoe/gofakeit

```
▶ ./bin/rand-cli
(main.Student) {
 Name: (string) (len=6) "PYRnUK",
 Age: (int) 3509420088055592154,
 Percentage: (float32) 2.5785265e+38,
 Height: (int) 6084305400485504554,
 Active: (bool) false,
 Meta: (main.Meta) {
  Description: (string) (len=10) "cOxtuWPLGR"
 }
}
Student{
        Name: "PYRnUK", Age: 3509420088055592154, Percentage: 2.5785265e+38,
        Height: 6084305400485504554,
        Meta: Meta{
                Description: "cOxtuWPLGR",
        },
}
```

main.go

```go
package main

import (
        "fmt"
        "os"
        "time"

        "github.com/brianvoe/gofakeit/v6"
        "github.com/davecgh/go-spew/spew"
        "github.com/hexops/valast"
)

type Student struct {
        Name       string
        Age        int
        Percentage float32
        Height     int
        Active     bool
        Meta
}

type Meta struct {
        Description string
}

func main() {
        seed := time.Now().Unix()
        faker := gofakeit.New(seed)

        var stu Student
        err := faker.Struct(&stu)
        if err != nil {
                fmt.Println(err)
                os.Exit(1)
        }

        spew.Dump(stu)
        fmt.Println(valast.String(stu))
}
```

Random Data Generation with struct field tags

➔ Generate meaningful data for the given fields

```
▶ ./bin/rand-cli
(main.Address) {
 Street: (string) (len=23) "34466 Crossroad chester",
 City: (string) (len=7) "Atlanta",
 Pincode: (string) (len=5) "12308",
 State: (string) (len=7) "Montana",
 Country: (string) (len=27) "Falkland Islands (Malvinas)"
}
Address{
        Street: "34466 Crossroad chester", City: "Atlanta",
        Pincode: "12308",
        State:   "Montana",
        Country: "Falkland Islands (Malvinas)",
}
```

main.go

```go
package main

import (
        "fmt"
        "os"
        "time"

        "github.com/brianvoe/gofakeit/v6"
        "github.com/davecgh/go-spew/spew"
        "github.com/hexops/valast"
)

type Address struct {
        Street  string `fake:"{street}"`
        City    string `fake:"{city}"`
        Pincode string `fake:"{zip}"`
        State   string `fake:"{state}"`
        Country string `fake:"{country}"`
}

func main() {
        seed := time.Now().Unix()
        faker := gofakeit.New(seed)

        var addr Address
        err := faker.Struct(&addr)
        if err != nil {
                fmt.Println(err)
                os.Exit(1)
        }

        spew.Dump(addr)
        fmt.Println(valast.String(addr))
}
```

Random Data Generation with functions

➔ Generate meaningful data for the given fields

```
▶ ./bin/rand-cli
(main.Address) {
 Street: (string) (len=20) "7800 East Lock shire",
 City: (string) (len=5) "Plano",
 Pincode: (string) (len=5) "67157",
 State: (string) (len=4) "Ohio",
 Country: (string) (len=5) "Sudan"
}
Address{
        Street: "7800 East Lock shire", City: "Plano",
        Pincode: "67157",
        State:   "Ohio",
        Country: "Sudan",
}
```

main.go

```go
package main

import (
        "fmt"
        "time"

        "github.com/brianvoe/gofakeit/v6"
        "github.com/davecgh/go-spew/spew"
        "github.com/hexops/valast"
)

type Address struct {
        Street  string
        City    string
        Pincode string
        State   string
        Country string
}

func main() {
        seed := time.Now().Unix()
        faker := gofakeit.New(seed)

        fakeAddr := faker.Address()

        addr := Address{
                Street:  fakeAddr.Street,
                City:    fakeAddr.City,
                Pincode: fakeAddr.Zip,
                State:   fakeAddr.State,
                Country: fakeAddr.Country,
        }

        spew.Dump(addr)
        fmt.Println(valast.String(addr))
}
```

Debugging help?

- Spew
  https://github.com/davecgh/go-spew
- valast
  https://github.com/hexops/valast
- Litter
  https://github.com/sanity-io/litter

```
spew.Dump(addr)
fmt.Println(valast.String(addr))
```

Generating embedded data

- ➔ `import _ "embed"`
- ➔ Use `//go:embed person.schema.json` comment to embed file above a variable `var personSchema []byte`
- ➔ Keep the `person.schema.json` file next to the cli `main.go` file

main.go

```go
import _ "embed"

//go:embed person.schema.json
var personSchema []byte
```

Generating embedded data (Demo)

main.go

```go
package main

import (
	_ "embed"
	"flag"
	"fmt"
	"os"
	"path/filepath"

	"rsc.io/getopt"

	"github.com/abhisekp/conf42-golang-2023-building-cli-app/internal/cmd"
)

//go:embed person.schema.json
var personSchema []byte

func GetCurrentDir() string {
	currDir, err := os.Getwd()
	if err != nil {
		return ""
	}
	return currDir
}

func main() {
	var genPersonSchema bool
	flag.BoolVar(&genPersonSchema, "generate", false, "Generate a person schema file")

	getopt.Aliases(
		"g", "generate",
	)

	getopt.Parse()

	if genPersonSchema {
		fullGeneratePath := filepath.Join(GetCurrentDir(), "person.schema.json")

		err := cmd.GeneratePersonSchemaToFile(personSchema, fullGeneratePath)
		if err != nil {
			_ = fmt.Errorf("failed to generate person schema: %w", err)
			return
		}
		fmt.Println("Generated file:", fullGeneratePath)
	}

}
```

```
▶ ./bin/rand-cli -g
Generated file: /home/abhisek/Projects/Personal/go/conf42-golang-2023-building-cli-app/person.schema.json
```

Generating multiple random data

➔ Pass the number of data to be generated from CLI arg

main.go

```go
package main

import (
	"flag"
	"fmt"

	"rsc.io/getopt"

	"github.com/abhisekp/conf42-golang-2023-building-cli-app/internal/cmd"
)

func main() {
	var numPersons int
	flag.IntVar(&numPersons, "num", 1, "Number of persons to generate")
	getopt.Aliases(
		"n", "num",
	)

	getopt.Parse()

	persons := cmd.GenPersons(numPersons)
	for _, p := range persons {
		fmt.Println(p)
	}
}
```

```
▶ ./bin/rand-cli -n 10
Forest Labadie (aged 46 yrs) lives in 8939 Points fort, Lubbock, Wisconsin, Mali
Lee Goldner (aged 98 yrs) lives in 74188 Square berg, Austin, Nevada, Austria
Asia Fahey (aged 93 yrs) lives in 524 West Run haven, Fresno, Arizona, Ghana
Einar Waelchi (aged 30 yrs) lives in 14596 Trafficway ton, Kansas , Hawaii, Belgium
Marjorie Veum (aged 15 yrs) lives in 95442 North Union bury, Toledo, Louisiana, Cuba
River Leuschke (aged 95 yrs) lives in 54631 Lake Ford bury, Minneapolis, Colorado, Somalia
Mazie Hettinger (aged 72 yrs) lives in 3197 New Forest side, Wichita, Vermont, Kiribati
Evans Howell (aged 36 yrs) lives in 55600 Port Avenue haven, Santa Ana, New Mexico, Curaçao
Jo Wilderman (aged 68 yrs) lives in 9006 South Dale bury, Greensboro, Mississippi, Lesotho
Elena Christiansen (aged 48 yrs) lives in 56619 Union stad, Mesa, Texas, Barbados
```

Generate data concurrently

➔ Use wait groups and go routines to generate concurrent data

```
▶ ./bin/rand-cli -n 10
Hermann Gleichner (aged 62 yrs) lives in 184 Ports view, San
Madie Hudson (aged 30 yrs) lives in 1205 North Ports borough
Payton Grimes (aged 16 yrs) lives in 8702 Knoll borough, Bak
Freddie Little (aged 14 yrs) lives in 6728 Lake Drive haven,
Ressie Spinka (aged 49 yrs) lives in 4532 North Motorway ber
Horace Ruecker (aged 8 yrs) lives in 78694 New Causeway mout
Billie Cronin (aged 60 yrs) lives in 474 Port side, Jersey ,
Amiya Crist (aged 28 yrs) lives in 178 South Tunnel mouth, S
Claudine Hammes (aged 36 yrs) lives in 3009 West Roads mouth
Lyric Larkin (aged 27 yrs) lives in 51876 West Crossing mout
```

person.go

```go
package cmd

import (
  "sync"

  "github.com/abhisekp/conf42-golang-2023-building-cli-app/pkg/person"
)

// GenPersons generates a `n` number of Person(s)
func GenPersons(n int) []*person.Person {
  persons := make([]*person.Person, n)
  wg := sync.WaitGroup{}
  for i := 0; i < n; i++ {
    wg.Add(1)
    go func(i int) {
      defer wg.Done()
      p := person.NewPerson()
      persons[i] = p
    }(i)
  }
  wg.Wait()
  return persons
}
```

Generate with “n” concurrency

➔ Use buffered channel to concurrency in go routines

```
▶ ./bin/rand-cli -n 10  -p
Duane Kozey (aged 88 yrs) lives in 307 Spurs fort, San Jose,
Ike Wilderman (aged 6 yrs) lives in 6257 New Stream fort, St.
Elfrieda Spinka (aged 43 yrs) lives in 64783 South Knoll berg
Quinten Harber (aged 48 yrs) lives in 3295 West Loop ton, Was
Jettie Cole (aged 3 yrs) lives in 229 Square mouth, San Franc
Camryn Wilkinson (aged 59 yrs) lives in 239 North Lock ton, W
Dagmar Quigley (aged 6 yrs) lives in 614 East Springs side, H
Christian Lang (aged 3 yrs) lives in 40106 Knoll view, Detroi
Korbin Jewess (aged 52 yrs) lives in 7177 Stravenue ton, Orla
Norene Grady (aged 28 yrs) lives in 921 New Islands ville, Al
Using concurrency of 8
```

person.go

```go
package cmd

import (
  "github.com/abhisekp/conf42-golang-2023-building-cli-app/pkg/person"
)

type GenPersonOptions struct {
  _           struct{}
  Concurrency int
}

var defaultOption = GenPersonOptions{
  Concurrency: 10,
}

// GenPersons generates a `n` number of Person(s)
func GenPersons(n int, options ...GenPersonOptions) []*person.Person {
  option := defaultOption

  if len(options) > 0 {
    option = options[0]
  }

  persons := make([]*person.Person, n)
  // Semaphore to limit the number of concurrent goroutines
  sem := make(chan int, option.Concurrency)

  for i := 0; i < n; i++ {
    sem <- i
    go func(i int) {
      p := person.NewPerson()
      persons[i] = p
      <-sem
    }(i)
  }

  // Wait for all goroutines to finish
  for i := 0; i < cap(sem); i++ {
    sem <- i
  }

  return persons
}
```

XML Output

➔ CLI flag for specifying output path and name

xml_data.go

```go
package cmd

import (
  _ "embed"
  "encoding/xml"
  "fmt"
  "io"
  "io/fs"
  "os"
  "path/filepath"
  "time"

  "github.com/abhisekp/conf42-golang-2023-building-cli-app/pkg/person"
  "github.com/abhisekp/conf42-golang-2023-building-cli-app/pkg/xml_data"
)

//go:embed output.xsd
var xmlSchema string

func GenXMLData(name string, persons []*person.Person) xml_data.Root {
  return xml_data.Root{
    XMLName: xml.Name{
      Local: "Persona",
    },
    XMLSchema:    "./output.xsd",
    XMLNamespace: "http://www.w3.org/2001/XMLSchema-instance",
    Name:         name,
    DateCreated:  time.Now().Format(time.RFC3339),
    Items:        persons,
  }
}
```

xml_data.go

```go
func GenXML(xmlData xml_data.Root, writer io.Writer) error {
  _, err := writer.Write([]byte(xml.Header))
  if err != nil {
    return err
  }

  enc := xml.NewEncoder(writer)
  enc.Indent("", "  ")

  if err := enc.Encode(xmlData); err != nil {
    fmt.Println("Error encoding XML:", err)
    return err
  }

  return nil
}
```

XML Schema Output

- ➔ Pass a new name to the XML
- ➔ Accepts `-c`, `-n`, `-o`, `-f`, and `-N` flags
- ➔ A `output.xsd` schema file will be created along with XML file which validates the generated xml.

```
Personal/go/conf42-golang-2023-building-cli-app  main ✔
▶ ./bin/xml-cli -n 10 -o tmp/ -N "Random Personas"
Using concurrency of 8
(miniconda3-latest)
Personal/go/conf42-golang-2023-building-cli-app  main ✔
▶ tree tmp
tmp
├── output.xsd
└── Random Personas.xml

0 directories, 2 files
```

xml_data.go

```go
func CreateSchemaFile(xmlFilepath string) error {
  basePath := filepath.Dir(xmlFilepath)
  schemaFilePath := filepath.Join(basePath, "output.xsd")
  file, err := os.Create(schemaFilePath)
  if err != nil {
    return err
  }

  defer func(file *os.File) {
    err := file.Close()
    if err != nil {
      fmt.Println("Error closing output file:", err)
      os.Exit(1)
    }
  }(file)

  _, err = file.WriteString(xmlSchema)
  if err != nil {
    return err
  }

  return nil
}
```

Prefilling with Pre-defined JSON Data

- ➔ Pass a new name to the XML
- ➔ Accepts `-c`, `-n`, `-o`, `-f`, `-i`, `-g`, and `-N` flags
- ➔ Use `-g` flag to generate `input.json` and `input.schema.json` files in `-o` specified directory
- ➔ Pass a `input.json` as input passing predefined data
- ➔ A `output.xsd` schema file will be created along with XML file which validates the generated xml

Prefilling with Pre-defined JSON Data (Demo)

Key Takeaways

- ➔ CLI apps are an important tool for developers, and Go is a great language for building them due to its strong support for command-line interfaces.
- ➔ The `flag` module in Go is a powerful tool for parsing command-line arguments and building CLI apps.
- ➔ To generate data concurrently, you can use Go's powerful concurrency features, such as channels and goroutines.
- ➔ You can override randomized data with predefined data.
- ➔ When building CLI apps in Go, it's important to keep the user experience in mind, and provide clear and helpful error messages when command-line arguments are incorrect.
- ➔ With the techniques covered in this talk, you'll be able to build powerful and flexible CLI apps in Go that can generate randomized data quickly and efficiently.

Resources

- 🌟 **Repository for the workshop**
  https://github.com/abhisekp/conf42-golang-2023-building-cli-app/commits/main
- 🎨 **Presentation for the workshop**
  https://speakerdeck.com/abhisekp/building-go-cli-app

- Awesome list of CLI apps
  https://github.com/agarrharr/awesome-cli-apps
- flag module
  https://pkg.go.dev/flag
- flag module Source Code
  https://cs.opensource.google/go/go/+/refs/heads/master:src/flag/flag.go;drc=f9cf2c4d0424e352f30d50b89d50eafbfb6fc019
- getopt library
  https://rsc.io/getopt
- Faker library
  https://github.com/brianvoe/gofakeit
- Cobra Framework for complex CLIs
  https://cobra.dev

THANKS

Do you have any questions?
Reach out to me...

contact@abhisekp.com

about.me/abhisekp

twitter.com/**abhisek**

linkedin.com/in/**abhisekp**

github.com/**abhisekp**

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, infographics & images by Freepik

Mascot and Slide Images are created using Midjourney AI

Code images are clickable links

Anonymous Feedback

Scan me

`https://to.absk.im/anon-feedback`