Fun times with Go

(the programming language)

Hello!

I am Maryum Styles

I am a software engineer at New Relic

Overview

1.

A brief history of Go

How and why did Go come about?

Golang: A history

- Created by three software engineers at Google
- Open source project in 2009
- Go version 1 released in 2012
- Go is currently on version 1.9
- Statically typed language
- Uses type inference
- Fun like Python and JS but more reliable!

Core OS

docker

kubernetes by Google

Reference:
https://golang.org/doc/faq#Origins

2.

Go basics

Let’s learn about strings and things!

Variables, Control Structures, and Maps
Oh my!

```go
// `var` declares 1 or more variables.
var a string = "initial"

// You can declare multiple variables at once.
var b, c int = 1, 2

// Go will infer the type of initialized variables.
var d = true

// Variables declared without a corresponding
// initialization are _zero-valued_. For example, the
// zero value for an `int` is `0`.
var e int

// The `:=` syntax is shorthand for declaring and
// initializing a variable, e.g. for
// `var f string = "short"` in this case.
f := "short"
```

Variables, Control Structures, and Maps
Oh my!

```go
// The most basic type, with a single condition.
i := 1
for i <= 3 {
    fmt.Println(i)
    i = i + 1
}

// A classic initial/condition/after `for` loop.
for j := 7; j <= 9; j++ {
    fmt.Println(j)
}

// `for` without a condition will loop repeatedly
// until you `break` out of the loop or `return` from
// the enclosing function.
for {
    fmt.Println("loop")
    break
}

// You can also `continue` to the next iteration of
// the loop.
for n := 0; n <= 5; n++ {
    if n%2 == 0 {
        continue
    }
    fmt.Println(n)
}
```

Variables, Control Structures, and Maps
Oh my!

```go
// To create an empty map, use the builtin `make`:
// `make(map[key-type]val-type)`.
m := make(map[string]int)

// Set key/value pairs using typical `name[key] = val`
// syntax.
m["k1"] = 7
m["k2"] = 13

// Printing a map with e.g. `Println` will show all of
// its key/value pairs.
fmt.Println("map:", m)

// Get a value for a key with `name[key]`.
v1 := m["k1"]
fmt.Println("v1: ", v1)

// The builtin `len` returns the number of key/value
// pairs when called on a map.
fmt.Println("len:", len(m))

// The builtin `delete` removes key/value pairs from
// a map.
delete(m, "k2")
fmt.Println("map:", m)
```

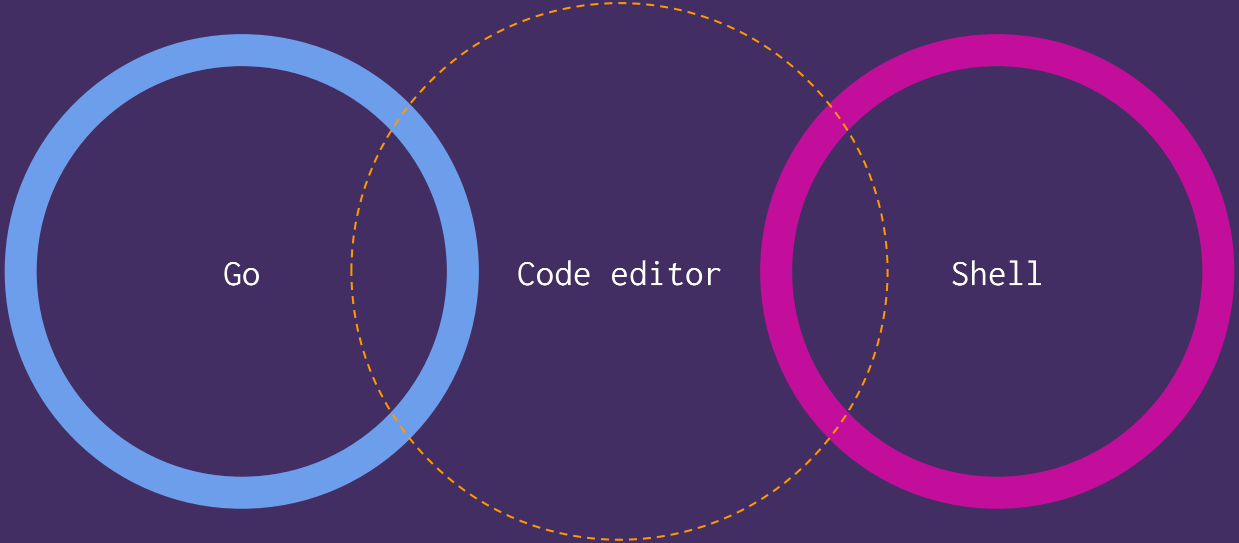
3.

Our mission

Should you choose to accept it

Image Manipulation

You're going to need



http://bit.ly/2DgcZdm

How can I tell if Go is installed?

```
go version
```

Something like “go version go1.8.3 darwin/amd64” should be returned

```
echo $GOPATH
```

This should return a location where you find go code

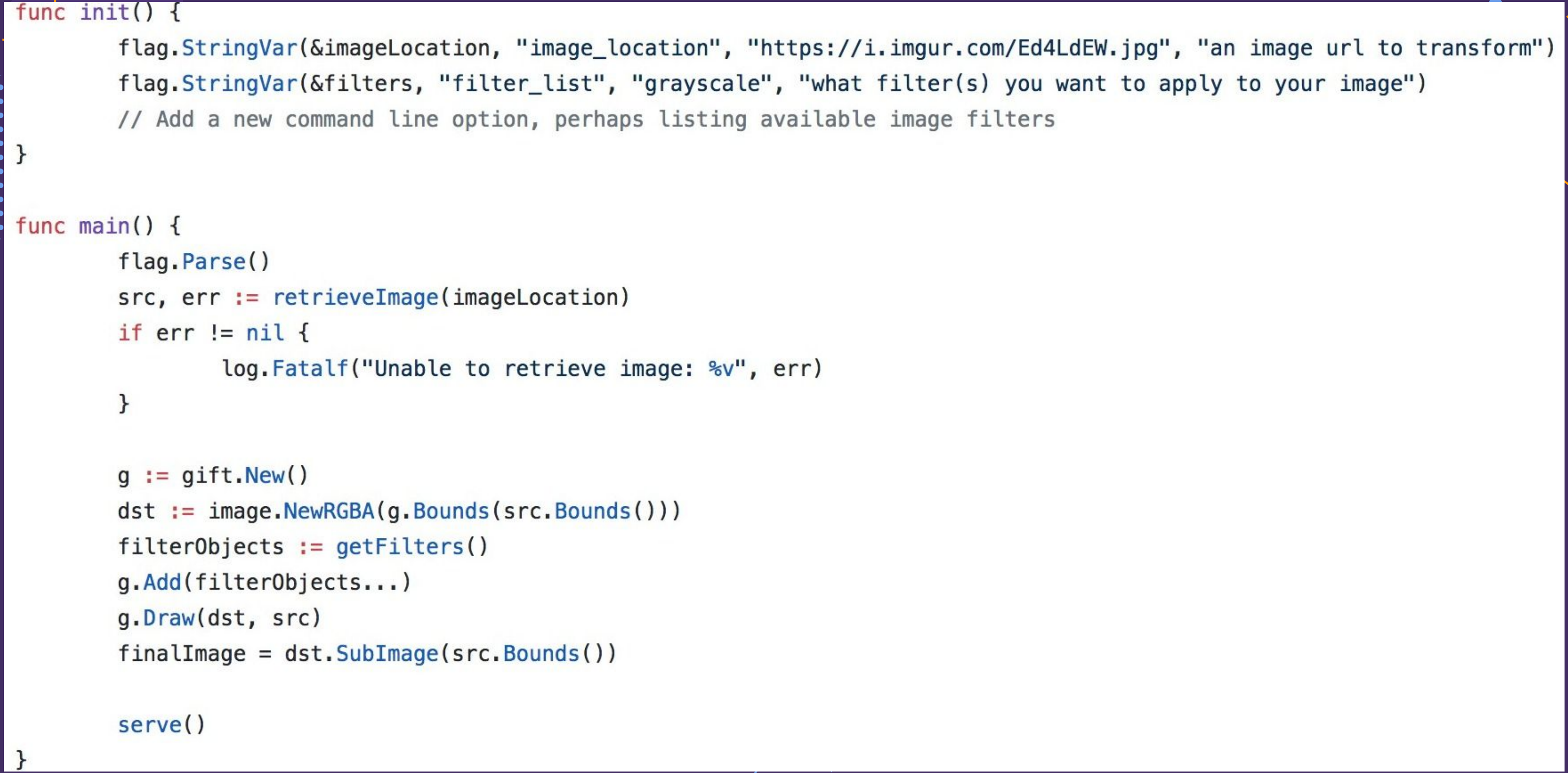
100%

You did it!

4.

Code walkthrough

Let’s get into the nitty gritty



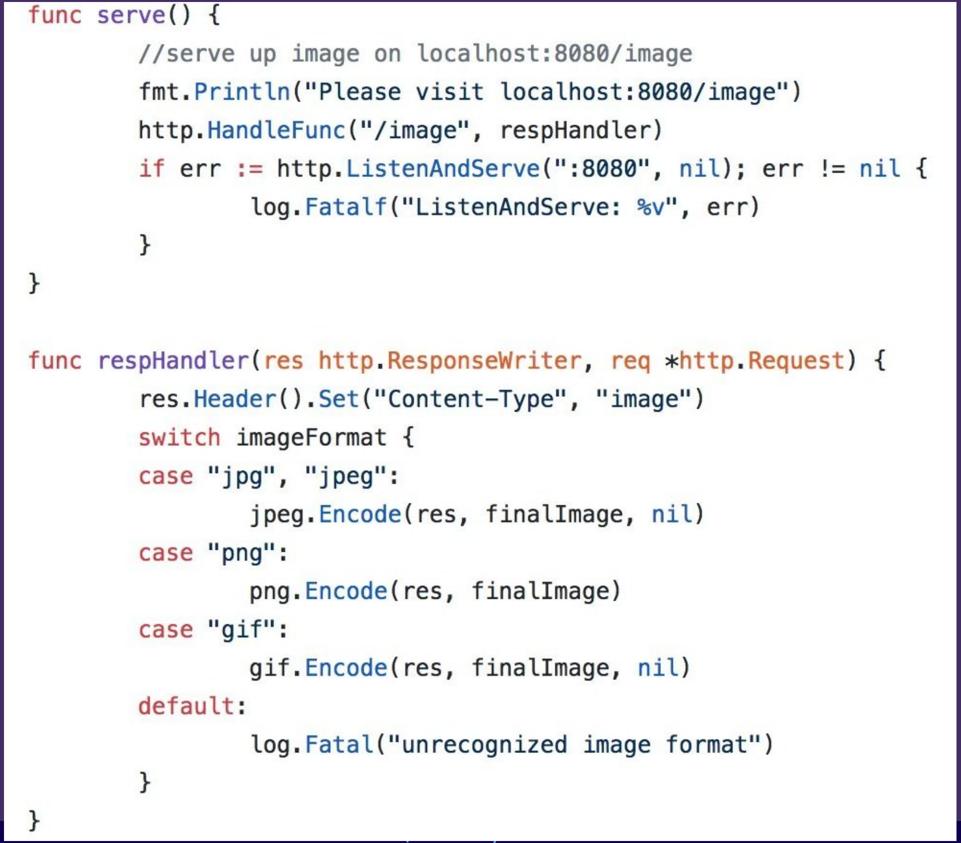
```go
func init() {
        flag.StringVar(&imageLocation, "image_location", "https://i.imgur.com/Ed4LdEW.jpg", "an image url to transform")
        flag.StringVar(&filters, "filter_list", "grayscale", "what filter(s) you want to apply to your image")
        // Add a new command line option, perhaps listing available image filters
}

func main() {
        flag.Parse()
        src, err := retrieveImage(imageLocation)
        if err != nil {
                log.Fatalf("Unable to retrieve image: %v", err)
        }

        g := gift.New()
        dst := image.NewRGBA(g.Bounds(src.Bounds()))
        filterObjects := getFilters()
        g.Add(filterObjects...)
        g.Draw(dst, src)
        finalImage = dst.SubImage(src.Bounds())

        serve()
}
```

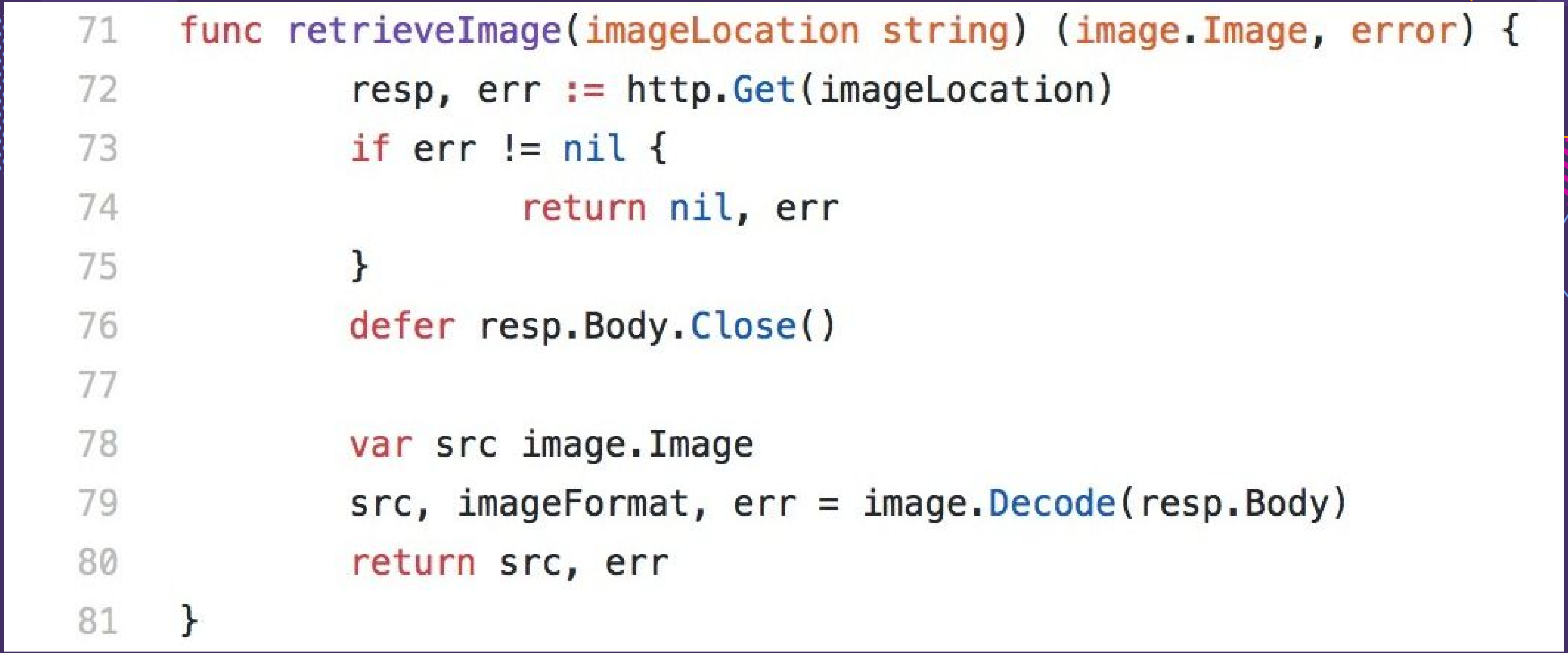
Init and Main



```go
func serve() {
    //serve up image on localhost:8080/image
    fmt.Println("Please visit localhost:8080/image")
    http.HandleFunc("/image", respHandler)
    if err := http.ListenAndServe(":8080", nil); err != nil {
        log.Fatalf("ListenAndServe: %v", err)
    }
}

func respHandler(res http.ResponseWriter, req *http.Request) {
    res.Header().Set("Content-Type", "image")
    switch imageFormat {
    case "jpg", "jpeg":
        jpeg.Encode(res, finalImage, nil)
    case "png":
        png.Encode(res, finalImage)
    case "gif":
        gif.Encode(res, finalImage, nil)
    default:
        log.Fatal("unrecognized image format")
    }
}
```

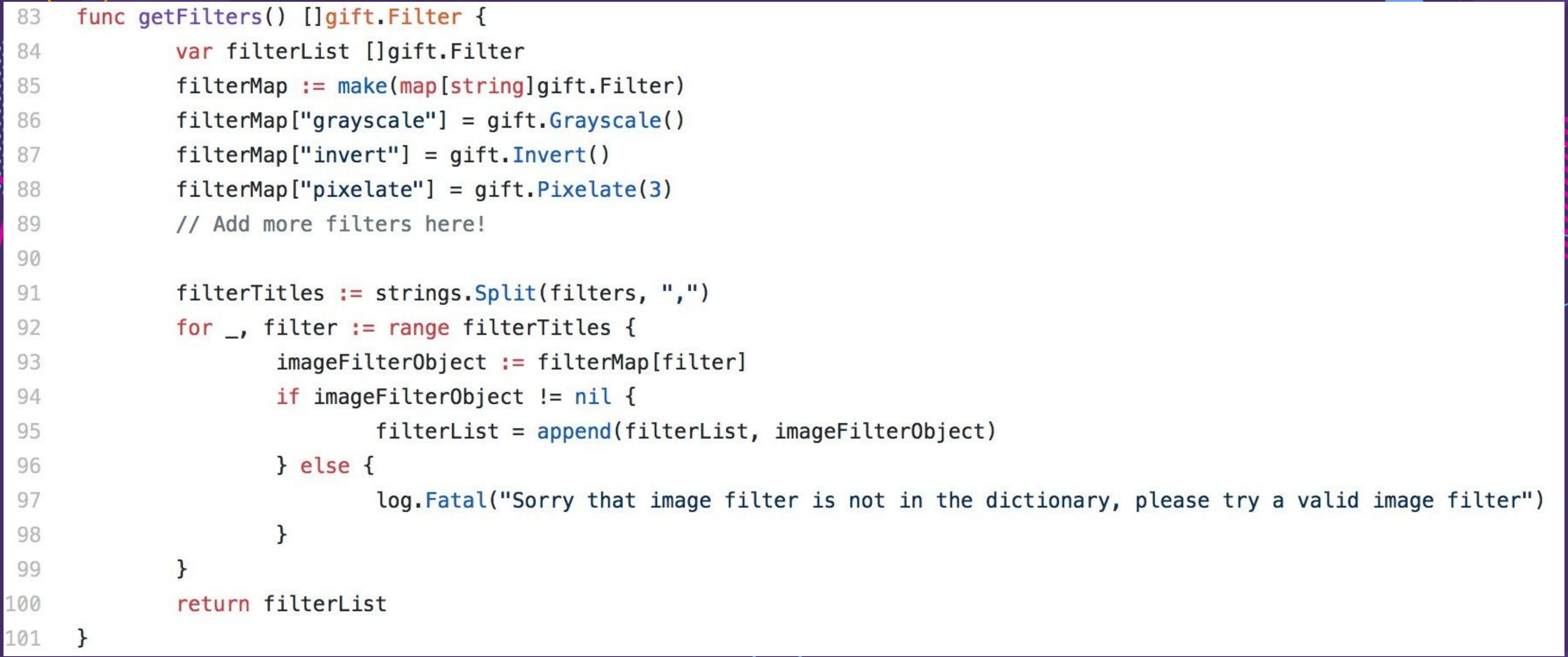
Http bits



```go
func retrieveImage(imageLocation string) (image.Image, error) {
	resp, err := http.Get(imageLocation)
	if err != nil {
		return nil, err
	}
	defer resp.Body.Close()

	var src image.Image
	src, imageFormat, err = image.Decode(resp.Body)
	return src, err
}
```

Make a request



```go
func getFilters() []gift.Filter {
	var filterList []gift.Filter
	filterMap := make(map[string]gift.Filter)
	filterMap["grayscale"] = gift.Grayscale()
	filterMap["invert"] = gift.Invert()
	filterMap["pixelate"] = gift.Pixelate(3)
	// Add more filters here!

	filterTitles := strings.Split(filters, ",")
	for _, filter := range filterTitles {
		imageFilterObject := filterMap[filter]
		if imageFilterObject != nil {
			filterList = append(filterList, imageFilterObject)
		} else {
			log.Fatal("Sorry that image filter is not in the dictionary, please try a valid image filter")
		}
	}
	return filterList
}
```

Hit em with that filter

5.

Compiling

Easier than you think!

```
go build .
```

This will create an executable in the code directory

```
./go-workshop
```

Run your executable

```
localhost:8080/image
```

Check it out!

Compile for another OS

```
GOOS=windows GOARCH= amd64 go build .
```

Go allows you to set variables that determine the OS and architecture for go build

GOOS -> android darwin dragonfly freebsd linux nacl netbsd openbsd plan9 solaris windows zos

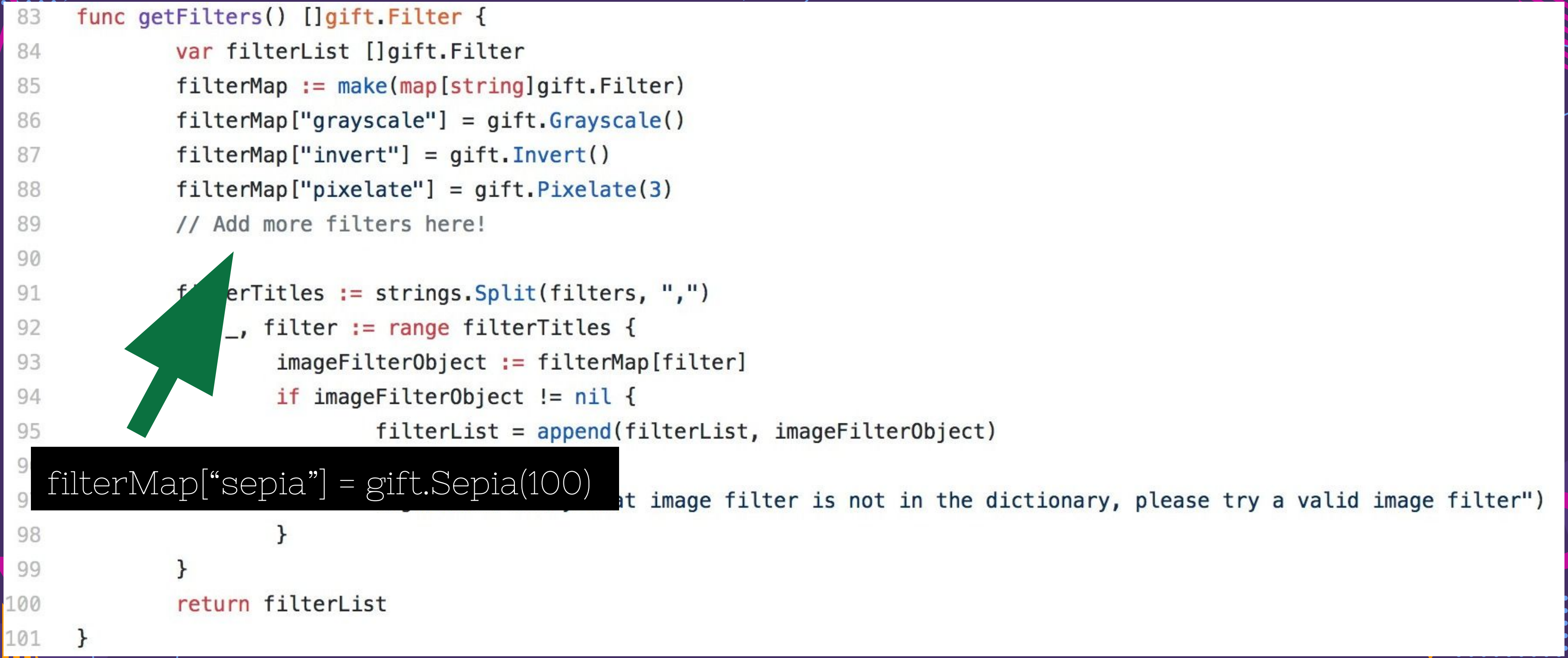
GOARCH -> 386 amd64 amd64p32 arm armbe arm64 arm64be ppc64 ppc64le mips mipsle mips64 mips64le mips64p32 mips64p32le ppc s390 s390x sparc sparc64

Reference:
https://github.com/golang/go/blob/master/src/go/build/syslist.go

6. Making changes

Let’s take a plunge in the deep end

Adding a filter



```
func getFilters() []gift.Filter {
        var filterList []gift.Filter
        filterMap := make(map[string]gift.Filter)
        filterMap["grayscale"] = gift.Grayscale()
        filterMap["invert"] = gift.Invert()
        filterMap["pixelate"] = gift.Pixelate(3)
        // Add more filters here!

        f    erTitles := strings.Split(filters, ",")
              _, filter := range filterTitles {
                imageFilterObject := filterMap[filter]
                if imageFilterObject != nil {
                        filterList = append(filterList, imageFilterObject)

                                  at image filter is not in the dictionary, please try a valid image filter")
                }
        }
        return filterList
}
```

filterMap["sepia"] = gift.Sepia(100)

7.

Tinkering

You have the basics, but play around some!

Tinkering ideas

Add more filters!

Save the image to a file

Change/add command line options

8.

Questions!

Resources

Learn Go

https://gobyexample.com/

Start a project

https://github.com/avelino/awesome-go