FROM RUBY TO GOLANG

A RUBY PROGRAMMER'S GUIDE TO LEARNING GO

JOEL BRYAN JULIANO

From Ruby to Golang

A Ruby Programmer's Guide to Learning Go

Joel Bryan Juliano

This book is for sale at http://leanpub.com/rb2go

This version was published on 2021-09-16

ISBN 1080944001, ISBN-13 978-1080944002

I dedicate this book to my loving, supportive and beautiful wife, and to my son.

Contents

Preface

Before I start, let me highlight what I love about Go and why I chose to learn it as my next programming language.

To begin with, let me say that Go hands-down is a great language; It is easy to read, with clear syntax. And it compiles to a single binary file which makes apps fast and compact and can compile to run on different platforms. And it's statically typed and garbage-collected, making it efficient.

It is like a modern C with package support, memory safety, automatic garbage collection, and concurrency baked-in. And you get all the nice features from a statically typed language, and IDEs love it, and so does your development workflow.

In today's world of cloud-native microservices, containerized architectures, You can be up-to-date with knowledge in Go. Many notable open-source projects are built using Go (i.e., Docker, Kubernetes, Etherium, and Terraform, to name a few). Those platforms have APIs and SDKs readily available natively for you to use. And many global companies have been using Go in production (i.e., Google, Netflix, Dropbox, Heroku, and Uber, to name a few), proving that it has been battle-tested and powerful mature language to based your work into.

This made me decide to pursue Go as my next language, and in 2018, I was hired by a company that uses Go. After 8+ years of working with Ruby, the first thing I did to learn Go was to relate to what I know in Ruby. And I thought it would be a good idea to document my learning process.

My initial intention was to keep it as my personal documentation and notes, but I decided to post it online after much careful thought and consideration. From one topic, it grew into a series of multiple online articles, discussing all my research and learnings about the language, collecting its analogies that I can compare with Ruby to help me learn it.

By teaching familiar implementations found in Ruby, you will see the correlation between the two languages, establishing familiar concepts to give you enough knowledge to be comfortable with Go and start programming with it.

This book was made with a Rubyist in mind in a friendly, conversational, and informal manner. All the learning metaphors are based on Ruby. I think it will help you learn the Go programming language when you already know Ruby. You don't need to take notes or remember things. Just continue reading and understand how things work from what you already know and how it relates and is applied in Go to learn the language.

This book does not discourage people from using Ruby. Ruby is still my favorite scripting language of all time, and I attest to Ruby's slogan, "A Programmer's Best Friend," because it is just a pleasure to use. I use Ruby all the time, and it has become a go-to extension of my mind to do something requiring computers. And I always get excited about new updates about Ruby.

Finally, I hope this book can help you get started, and master Go as your next programming language. With so many great opportunities out there to build great things with Go, you will not regret learning it.

Now, let's get started!

Introduction

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

About the Author

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Chapter 0: Go Primer

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Package Name and Imports

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Println

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Printf

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Sprintf

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Function Basics

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Function Parameters and Arguments

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Function Return Types

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Main Function

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Chapter 1: Structs

Everything needs communication. Animals and humans communicate using physical gestures and sounds. Most insects have a special antenna in them that they use to communicate to other insects or emits sounds or scents to exchange information with other animals. Plants also have a special way of communicating with other plants through electric signaling or using other hosts to send their messages.

From gigantic animals to small microscopic bacterias, viruses to cells, all have a special way to exchange messages. They all have a way to coordinate information with each other.

Computers and software also require much communication.

Inside the computer hardware, electrical signals are sent to each component where they are task to perform a particular function. And computers themselves communicate to other computers via wired, wireless, and The Internet.

It is hard to imagine a world without communication!

In programming, there are also various levels of ways to communicate your code. In this chapter, we will discuss ways we can communicate to individual parts of code.

If you need to pass, exchange, and coordinate values bound to specific parts of the code, we can use a variable.

Variables can either be a local variable that you can only access inside the function or a global variable that you can globally access throughout your code.

For our first program example, I assume that you have prior Ruby knowledge, so that we will do a simple recap in Ruby. We will create a function that takes a single word input to dissect it in Go. This code will output the word "Hello World."

Here, we have a local variable in Ruby. In this example, we declared a local variable `hello` inside a function `say_hello`.

```ruby
def say_hello(message)
  hello = "Hello"
  puts hello + message
end

say_hello("World")
```

And when we run the Ruby code, it will output "Hello World."

```
$ ruby say_hello.rb

Hello World
```

However, if we want to change the `hello` variable to store a different message, we cannot directly change it from outside the function.

```
def say_hello(message)
  hello = "Hello"
  puts hello + message
end

hello = "Hi"
say_hello("World")
```

So even though we change the `hello` variable, this will still output a "Hello World."

```
$ ruby say_hello.rb

Hello World
```

Questions

1. What have we learned?
We learned to use a local variable to store a value that we can only access inside the function.

2. Why do we need communication?
We need communication to exchange information.

3. What is a variable?
A variable is a name that you can use to store a value.

4. Why do you use a variable?
You use a variable to store a value that you can use later in your code.

5. How do you use a variable?
You use a variable by declaring it first, then you can assign a value to it.

6. How do we pass and coordinate information in our code?
We pass and coordinate information in our code by using variables.

7. What is a local variable?
A local variable is a variable that you can only access from inside the function.

8. What is a global variable?
A global variable is a variable that you can access from anywhere in your code.

Instance Variables

How do we change the information from inside different functions?

What we need is an instance variable. An instance variable is a variable that you can access via a single reference. In Ruby, the instance variable starts with an `@` sign, followed by the variable name.

You can treat an instance variable like any normal variable, only that you can pass or modify information around it to other functions regardless of its location.

So back to our example, we changed the local variable `hello` to `@hello`, and then we can change the message of the variable.

```
1  def say_hello(message)
2    @hello = "Hello"
3    puts @hello + message
4  end
5
6  @hello = "Hi"
7  say_hello("World")
```

This will output a "Hi World" message.

```
1  $ ruby say_hello.rb
2
3  Hi World
```

This is very convenient because we can pass values to other functions without initializing any object.

Questions

1. What have we learned?
We learned that we could pass values to other functions without initializing an object using instance variables.

2. Why is it useful?
It is useful because it is used to store data that is specific to an instance of a class or stores a value that is specific to a particular object.

3. What is an instance variable?
An instance variable is a variable that is associated with an instance of a class that you can access via a single reference.

4. How do you create an instance variable in Ruby?
You can create an instance variable by using the `@` sign, followed by the variable name.

Struct

`struct` is a **type** that you can declaratively group and define data fields accessible via a single reference. `struct` also provides a way to pass a value from within functions.

In Go, we can use structs to store a value from our functions. Here is an example of how to create a struct.

```
type Employee struct {
  FirstName string
  LastName  string
}
```

Using a struct, you can set a value and get a value from within your functions. This capability to exchange data throughout your code from within functions is useful for exchanging and coordinating data onto other parts of your code.

In Ruby, a struct can be equivalent to an **instance variable**.

In the following example, let's discuss this in detail, starting with a Ruby implementation on how you pass values from within different parts of the code. We can proceed with a Go example on how to pass values from one function to another function.

In the following Ruby example, a class `Dog` has an initializer that accepts a parameter `breed` and sets that parameter to an instance variable `@breed`, which provides data access from within the class.

In this case, a public method `kind` directly returns the value passed to the initializer.

Ruby Instance Variables

```
class Dog
  def initialize(breed)
    @breed = breed
  end

  def kind
    @breed
  end
end

dog = Dog.new('Rottweiler')
dog.kind
```

```
$ ruby dog.rb

Rottweiler
```

The use of instance variables in the Ruby example can be rewritten in Go, using `struct`.

In the next example, a `dog` struct defines a property `breed` with a value type `string` and inside `main()`. The struct `dog` initializes to a variable `kind` with its property `breed` filled in.

Go Private Struct

```
package main

import "fmt"

type dog struct {
  breed string
}

func main() {
  kind := dog{
    breed: "Rottweiler",
  }

  fmt.Println(kind.breed)
}
```

```
$ go run dog.go

Rottweiler
```

Since `struct` can group similar data field types, you can extend this struct by adding multiple fields.

```
type dog struct {
  name  string
  breed string
  age   int
}
```

Then we can access multiple fields this way.

Go Private Struct

```go
package main

import "fmt"

type dog struct {
  name  string
  breed string
  age   int
}

func main() {
  pet := dog{
    name: "Maximus",
    breed: "Rottweiler",
    age: 5,
  }

  fmt.Printf("%+v", pet)
}
```

```
$ go run dog.go

{name:Maximus breed:Rottweiler age:5}
```

Questions

1. What have we learned?
We learned that struct is a type that you can declaratively group and define data fields accessible via a single reference.

2. Why is it useful?
Struct is useful because it provides a way to pass a value from within functions.

3. What is a struct?
A struct is a user-defined type used to group many variables of different types.

4. What is the equivalent of a struct in Ruby?
An instance variable.

5. How do you declare a variable in Go?
You declare a variable in Go by using the keyword var.

6. How do you create a struct in Go?
You can declare a struct using the `struct` *keyword, followed by the struct name, field name, and type.*

7. What is a private struct in Go?
A private struct is a struct that is only visible to the package it is defined in.

8. What is a struct field in Go?
A struct field is a variable that is part of a struct.

Public Structs

Take note that the struct `dog` is only available internally from within the package.

To make this public, the struct type alias name needs capitalization, in this case to `Dog`.

The same naming mechanics can also apply to the *struct field names*, *functions*, and *variables*. Provide an option to make its resource publicly available for other packages by capitally them.

Go Public Struct

```
1  package main
2
3  import "fmt"
4
5  type Dog struct {
6    Breed string
7  }
8
```

```
func main() {
  kind := Dog{
    Breed: "Rottweiler",
  }

  fmt.Println(kind.Breed)
}
```

```
$ go run dog.go

Rottweiler
```

In most cases, the equivalent functionality of an *instance variables* in Go would be using `struct`. And you can extend `struct` by attaching it to a function, forming methods to a struct that you can perform *mutations* of existing values.

Questions

1. What have we learned?
Structs are a way to group data, and you can also make it publicly available for other packages.

2. Why is it useful?
Sharing packages in Go is useful because it allows you to reuse code in different projects.

3. How do you make your Golang struct publicly available?
You can make your struct publicly available by capitalizing the struct type alias name.

4. What is another Golang functionality that you can make public?
You can also make your functions and variables publicly available by capitalizing their names.

5. How do you mutate an existing struct value?
By attaching a struct to a function, you can perform mutations of existing values.

Attaching a Struct to a Function

Supposed we like to perform a method on a declared variable.

In Ruby, this is an OOP call since Ruby can create methods in a class. This allows you to call those methods when you initialize the class using dot notation.

However, Go's first-class citizens are functions and not an OOP language by design. Since there's no concept of class, how do we attach a method from within a declared variable in a similar manner in Ruby?

In Go, we can associate a struct to a function by specifying the type of that function as one of the declared structs. This will provide the capability to create methods from within functions.

In the following example in Ruby, an instance variable `produce`
declares as a receiver. There are also three methods to operate on the variable: ‘ add_item, change_-item `and` items‘.

In the `add_item` method, using a double-splat[1] parameter operator, we generate a hash from the method appending the values to `produce`.

The `change_item` method deletes the entry from the
hash and then appends to `produce` as a new entry.

Operating On A Reciever Instance Variable in Ruby

```
class Basket
  def initialize
    @produce = []
  end

  def add_item(**entry)
    # Raise an error if name, flavour and
    # kind keys are not passed
    items = %w[name flavour kind]

    unless items.any? { |key| entry.key? key.to_sym }
      raise "Usage: add_item(name:    '..',
                             flavour: '..',
                             kind:    '..')"
    end

    @produce << entry

    puts "Entry #{entry[:name]} created!"
  end

  def change_item(name, entry)
    # Delete existing record
    item = @produce
           .delete_if { |h| h[:name] == name }
           .first
```

[1] https://ruby-doc.org/core-2.3.0/doc/syntax/calling_methods_rdoc.html#label-Hash+to+Keyword+Arguments+Conversion

```
27              .dup
28
29      item = entry
30
31      # Add it to the instance variable
32      @produce << item
33      @produce.uniq!
34
35      puts "Item #{name} changed!"
36    end
37
38    def items
39      puts "There are #{@produce.count} number of items in
40  the basket"
41
42      @produce.each do |entry|
43        item(entry)
44      end
45    end
46
47    private
48
49    def item(entry)
50      puts "Name: #{entry[:name]}"
51      puts "Flavour: #{entry[:flavour]}"
52      puts "Kind: #{entry[:kind]}"
53    end
54  end
55
56  basket = Basket.new
57
58  basket.add_item(
59    name: 'apple',
60    flavour: "It's a little sour and bitter, but mostly
61  sweet, not at all salty, very juicy in general",
62    kind: 'fruit'
63  )
64
65  basket.add_item(
66    name: 'carrot',
67    flavour: '',
68    kind: 'veggies'
69  )
```

```
70
71 basket.change_item(
72   'carrot',
73   name: 'cucumber',
74   flavour: 'Slightly bitter with a mild melon aroma,
75 and planty flavor.',
76   kind: 'veggies'
77 )
78
79 basket.items
```

```
1  $ ruby basket.rb
2
3  Entry apple created!
4  Entry carrot created!
5  Item carrot changed!
6
7  There is 2 number of items in the basket.
8
9  Name: apple
10 Flavor: It's a little sour and bitter, but slightly sweet,
11 not at all salty, juicy in general
12 Kind: fruit
13
14 Name: cucumber
15 Flavor: Slightly bitter with a mild melon aroma, and
16 leafy flavor.
17 Kind: veggies
```

To rewrite Go, a struct must represent the `produce` with properties `name`, `flavor` and `kind`.

Another struct named `basket` will be an array of `produce` since `basket` is an array of a struct that will contain the `produce` struct.

We will define a function and attach it to `basket` to interact with `produce`. We will call the defined `basket` functions as `add_item`, `change_item`, and `items`.

Operating on a Receiver Struct in Go

```go
package main

import "fmt"

type basket []produce

type produce struct {
        name    string
        flavour string
        kind    string
}

func (p *basket) add_item(entry produce) {
        *p = append(*p, entry)

        fmt.Printf("Entry %s created!\n", entry.name)
}

func (p basket) change_item(name string, entry produce) {
        for key, val := range p {
                if val.name == name {
                        p[key] = entry
                }
        }

        fmt.Printf("Item %s changed!\n", name)
}

func (p basket) items() {
        fmt.Printf("There are %d number of items in the basket\n",
                len(p))

        for _, val := range p {
                fmt.Printf(`Name: %s\n
                Flavour: %s\n
                Kind: %s\n`,
                        val.name,
                        val.flavour,
                        val.kind)
                fmt.Println("")
        }
}
```

```
43
44  func main() {
45          basket := new(basket)
46
47          basket.add_item(
48                  produce{
49                          name: "apple",
50                          flavour: `It's a little sour and bitter, but mostly
51  sweet, not at all salty, very juicy in general.`,
52                          kind: "fruit",
53                  },
54          )
55
56          basket.add_item(
57                  produce{
58                          name:    "carrot",
59                          flavour: "",
60                          kind:    "veggies",
61                  },
62          )
63
64          basket.change_item("carrot",
65                  produce{
66                          name: "cucumber",
67                          flavour: `Slightly bitter with a mild melon aroma
68  and planty flavor.`,
69                          kind: "veggies",
70                  },
71          )
72
73          basket.items()
74  }
```

```
$ go run basket.go

Entry apple created!
Entry carrot created!
Item carrot changed!

There is 2 number of items in the basket.

Name: apple
Flavor: It's a little sour and bitter, but slightly sweet,
not at all salty, juicy in general.
Kind: fruit

Name: cucumber
Flavor: Slightly bitter with a mild melon aroma and leafy
flavor.
Kind: veggies
```

Questions

1. What have we learned?
We learned that we could attach a struct to a function in Go.

2. Why is it useful?
It's a way to pass data to a function.

3. How do you create a function with a struct in Go?
You can create functions with a struct by specifying the type of the function as one of the declared structs.

4. Is Golang an OOP language?
*No, Go is not an OOP language.

5. What is a double-splat parameter operator?
The double-splat operator is a way to pass multiple arguments to a function.

6. How do you create a double-splat parameter operator in Ruby?
*In Ruby, we can create a splat functionality by using the *args and **kwargs*

Pass-by-value and Pass-by-reference

In our above basket example, notice two types of functions attached to `basket`?

It might be a new concept coming from Ruby, but when dealing with pointers and references, there is two semantics of passing a value to a variable.

Let's discuss them in detail.

Those are the two types of parameter passing in Go, namely *pointer receiver* and *value receiver*.

- *Value receivers* are *pass-by-value*, which means that the variable will use an actual value or a resulting value.
- *Pointer receivers* are *pass-by-reference*, which takes in the memory address pointing to the value and passes it to the variable.

Ruby by default is pass-by-value, and in Go, we can use those two types of semantics of passing and using a variable value. We will discuss them further in detail.

When deciding the proper way of passing variables, the Go community had created documentation on utilizing the proper semantics between pointer receivers and value receivers. See the official Golang code review comment[2] for receiver types.

Questions

1. What have we learned?
We learned that there are two types of the semantics of passing variables in Go.

2. Why is it useful?
It is useful because it helps us understand the difference between the two types of semantics of passing variables.

3. What is pass-by-value?
Pass-by-value means that the value of the variable is passed to the function.

4. What is pass-by-reference?
Pass-by-reference means that the address of the variable is passed to the function.

5. Is Ruby a pass-by-value or pass-by-reference?
It's pass-by-value.

6. Is there a proper semantic in between pointer receivers and value receivers in Golang?
Yes, there is a community-supported semantic in between pointer receivers and value receivers in Golang.

[2] https://github.com/golang/go/wiki/CodeReviewComments#receiver-type

Pointer Receiver

Pointer receiver is pass-by-reference, which means that we pass the reference to the memory address of the resulting value to a variable.

Pointer receivers will create a copy to a new variable referencing the value's memory address per each assignment. When you modify the parameter values, it will be a modification referencing the memory address of the original variable's value.

To explain this further, let's go back to our previous example. In the previous example, `add_item` expects a `produce` argument, and the receiver of this function is `basket`.

Notice that there is an asterisk before the receiver type, `p *basket`, which means that the receiver `basket` can be *accessed* and *mutated* directly within the function.

Pointer receivers are both setters and getters in Go, in the sense that the receiver `basket` is settable and gettable directly.

Pointer receivers can be an ideal use case when you are looking for an `attr_accessor` analog in Go, compared to the same behavior keen in Ruby. Here's the general form of constructing a pointer receiver.

Pointer Receiver in Go

```
func (receiverName *receiverType)
     funcName(paramName paramType) {
  // we can directly set the value of the
  // receiver type (setter)
  *receiverName = paramName

  // we can also access the value of the
  // receiver type (getter)
  fmt.Println(receiverName)
}
```

Questions

1. What have we learned?
Pointer receivers are a way to pass a pointer to a function.

2. Why is it useful?
It is useful because the function can use the pointer to access the data.

3. Is the pointer receiver a pass-by-reference or pass-by-value?
Pointer receivers are pass-by-reference.

4. What is an equivalent attribute accessor in Golang?
Pointer receivers are an equivalent of attribute accessors in Go.

5. What is a pointer-receiver in Go?
A pointer-receiver is a function that takes a pointer as an argument.

6. Why there's an asterisk before the receiver type?
The asterisk before the receiver type is a pointer receiver, which means it can be accessed and mutated directly.

Value Receiver

Value receiver is pass-by-value, which means that we pass the actual value or a resulting value to a variable. Value receivers will create a new independent variable copying the original value per each assignment.

Two functions in our previous example utilize a value receiver function: those methods where the receiver type does not start with an asterisk, namely, the `change_item` and `items` functions which are also a method for the `basket` struct.

You might ask: *Why does `change_item` mutate the value of the `basket` struct? Does this mean that the value receiver is also a setter? And the same with the pointer receiver?*

You can modify the parameter values, but changes are not forwarded to the original variable. It is possible to modify existing variables with records, and value receivers can be setters of an initialized variable. However, any modifications to the variable will not reflect on new struct records. We must create the struct first and initialize it separately before passing values to the new structure record.

Value receivers are ideal for concurrent applications because they do not modify the original reference but create a new reference copy, making value receivers thread-safe. To explain this much further, let's go back to our example above.

In our pointer receiver example, we used `append` to append a `produce` in a newly initialized `basket` array. The `change_item` mutates an already initialized `basket` variable but cannot mutate the struct basket directly. Here's the general form of creating a value receiver.

Value Receiver in Go

```
func (receiverName receiverType)
     funcName(paramName paramType) {
  // we can set the value of an
  // _already initialised_ receiver type
  // but we cannot modify the receiver
  // type directly
  receiverName = parameterName

  // we can access the value of the
  // _already initialised_
  // receiver type (getters)
  fmt.Println(receiverName)
}
```

Questions

1. What have we learned?
Value receivers are a way to pass a value to a function and creates a new copy of the original value

2. Why is it useful?
They are used to pass a value to a function and not a reference to the value.

3. What is a value-receiver in Ruby?
A value-receiver is a class with a method that accepts a value of another class as a parameter.

4. What is a value-receiver in Golang?
A value-receiver is a function that receives a value as an argument.

5. Are the value receiver the same as the pointer receiver?
No, value receivers are not the same as pointer receivers. Value receivers are pass-by-value, which means that we pass the actual value or a resulting value to a variable. Value receivers will create a new independent variable copying the original value per each assignment.

6. In value receivers, will any modifications to the variable reflect on new struct records?
No, any modifications to the variable will not reflect on new struct records. We still need to create the struct first, initialize it separately before we can pass values to the new struct record.

7. Are value receivers thread-safe?
Yes, value receivers are ideal for concurrent applications because they do not modify the original reference but create a new reference copy, making value receivers thread-safe.

Decouple and Reuse Structs through Inheritance

One of the nicest features of `struct` is decoupling the data structures into smaller chunks inheriting a common structure, allowing specific implementations for each data structure.

Modularizing your data structure is a good practice by separately grouping your data struct. When decoupling your structs into smaller chunks, you can maximize the reusability of your struct because it
allows you to interchange your struct into varied use cases, focusing on the content of each structure.

In the long run, your code will be easy to maintain due to the
modularized organization of your structures, adding the overall
the simplicity of using and maintenance of your code.

In the following example, we create a common `Animal` struct and a `Dog` struct. Specifying the name of the struct on top of the field inherits the struct and its field properties. In this case, the Dog struct inherits the Animal struct and its fields. Adding an asterisk before the name makes the inheritance a pointer receiver, allowing modifications to the struct field values.

```
type Animal struct {
  Kind    string
  Habitat string
  Origin  string
  Diet    string
}

type Dog struct {
  *Animal
  Name  string
  Breed string
}
```

And we can also have the ability to inherit multiple structures. Here's the complete example where the `Dog` and `Cat` structs are inheriting both `Animal` and `Owner`, and in the `main()`, we can add the values for each field.

```go
package main

import "fmt"

type Animal struct {
  Kind string
  Diet string
}

type Owner struct {
  Name    string
  Country string
}

type Dog struct {
  *Animal
  *Owner
  Name  string
  Breed string
}

type Cat struct {
  *Animal
  *Owner
  Name  string
  Breed string
}

func main() {
  var dog Dog

  dog.Name = "Maximus"

  dog.Animal = &Animal{
    Kind: "Dog",
    Diet: "Omnivorous",
  }

  dog.Breed = "Pitbull"

  dog.Owner = &Owner{
    Name:    "John",
    Country: "USA",
```

```
  }

  fmt.Printf("Name of %s is %s\n", dog.Animal.Kind,
dog.Name)
  fmt.Printf("%s is a %s with an %s diet\n", dog.Name,
dog.Breed, dog.Animal.Diet)
  fmt.Printf("%s is owned by %s who lives in %s\n",
dog.Name, dog.Owner.Name, dog.Owner.Country)
}
```

```
$ go run animal.go

The name of Dog is Maximus.
Maximus is a Pitbull with an Omnivorous diet
Maximus is owned by John, who lives in the USA
```

Questions

1. What have we learned?
We learned that structs are a great way to organize your data, and you can use inheritance to create a common structure and then create specific structs that inherit the common structure.

2. Why is it useful?
It's useful because it allows you to create a common structure that you can use in multiple structs, and it allows you to create specific structs that inherit the common structure.

3. Can you decouple data structures into a struct?
Yes, you can decouple data structures in the struct.

4. Why do you want to decouple your structs into smaller chunks?
By decoupling your structs in smaller chunks, you can maximize the reusability of your struct because it allows you to interchange your struct into varied use cases, focusing on the content of each structure.

5. Why is modularization a good practice?
In the long run, your code will be easy to maintain, due to the modularized organization of your structures, adding the overall simplicity of using and maintenance of your code.

6. What does adding an asterisk before a name in a struct field do?
Adding an asterisk before the name makes the inheritance a pointer receiver, allowing modifications to the struct field values.

Anonymous Structs

An anonymous struct is a struct you can declare and initialize on the fly without explicit declaration via `type`. It's declared in an inline manner along with your code, which provides a flexible way of declaration and usage of your data structures.

From a developer's perspective, this provides speedy invocations of your data structures when you need it because anonymous structs can immediately invoke your data structures as needed along with your code. Other cases where anonymous structs are useful are when a name is not needed for the operation.

Anonymous struct also provides a way to avoid needless declarations of type name alias, which can cause namespace pollution for your struct. However, it's recommended to avoid deep nested anonymous structs due to the risk of unreadable code. This may result in a code that is hard to maintain.

Finally, anonymous structs is a good and cheap alternative to an empty `interface{}` type.

```
package main

import (
  "fmt"
)

func main() {
  Animal := struct {
    Kind string
    Diet string
  }{"Dog", "Omnivorous"}

  fmt.Println(Animal.Kind, "-", Animal.Diet)
}
```

```
$ go run anonymous_struct.go

Dog - Omnivorous
```

Questions

1. What have we learned?
We learned that we could declare and initialize a struct on the fly without explicit declaration via type.

2. Why is it useful?
Anonymous structs are useful when a name is not needed for the operation.

3. What are anonymous structs?
Anonymous structs are structures you can declare and initialize on the fly without explicit declaration via `type`*. It's declared in an inline manner along with your code.*

4. What is the advantage of anonymous structs for a developer?
Anonymous structs provide speedy invocations of your data structures when you need them because anonymous structs can immediately invoke your data structures as needed along with your code. Other cases where anonymous structs are useful are when a name is not needed for the operation.

5. What problem do anonymous structs solve?
Anonymous structs provides a way to avoid needless declarations of type name alias, which can cause namespace pollution for your struct.

6. What are the risks of using anonymous structs?
Anonymous structs are a good and cheap alternative to an empty `interface{}` *type. However, it's recommended to avoid deep nested anonymous structs due to the risk of unreadable code. This may result in a code that is hard to maintain.*

Anonymous Struct Fields

In OOP languages, the properties of an object are collectively called attributes. Since Go is not an OOP language, attribute names in an OOP property are called fields.

Normally, a struct has a declared field name with an attached type. However, this is not always the case. You can also create a struct without field names, leaving only its type.

The way to access the value of an anonymous field is by calling the literal name of the field type of the struct, for example:

```go
package main

import (
  "fmt"
)

func main() {
  animals := struct {
    int
    string
  }{1, "Dog"}

  fmt.Println(animals.int)
  fmt.Println(animals.string)
}
```

```
$ go run anonymous_struct_fields.go

1
Dog
```

You can only access the value by calling the literal name of the type in the struct, and with this semantics, it comes with a limitation.

It is important to note that when using anonymous field names, you have a limitation of having a declaration of the same type only once since anonymous struct fields need to have unique value types. Otherwise, there cannot be a distinction between each type.

Questions

1. What have we learned?
Anonymous struct fields are useful when you want to create a struct with a unique type, and you don't want to create a new type just for that.

2. Why is it useful?
Go's anonymous struct fields are a great way to pass data to functions without creating a new type.

3. What are anonymous struct fields?
Anonymous struct fields are struct fields without a name.

4. What are the limitations of anonymous struct fields?
Anonymous structs are limited to a single type declaration.

5. What is a property of an object in OOP?
In OOP, properties of an object are collectively called attributes.

6. What is the equivalent object attribute in Go?
An attribute is a property of an object. In Go, an attribute is a field of a struct.

7. How do you access the value of an anonymous field?
You can only access the value by calling the literal name of the type in the struct, and with this semantics, it comes with a limitation.

8. What is the limitation of an anonymous field?
It is important to note that when using anonymous field names, you have a limitation of having a declaration of the same type only once since anonymous struct fields need to have unique value types. Otherwise, there cannot be a distinction between each type.

Chapter Questions

1. How do you create a function in Go?
You use the keyword `func` followed by the function name, a list of parameters in parentheses, and the function body.

2. How do you create a variable in Go?
You use the keyword var followed by the variable name and the type.

3. How do you create a pointer in Go?
*You use the keyword * followed by the type of the pointer and the variable name.*

4. What is a struct?
A struct is a collection of fields.

5. What is a field?
A field is a variable that is part of a struct.

6. How do you create a struct in Go?
You use the keyword struct followed by the struct name, a list of fields in curly braces, and the body.

7. What does capitalizing a struct, variable, and function do?
It makes it publicly accessible outside the package.

8. What is an anonymous struct?
An anonymous struct is a struct without a name.

9. Why use an anonymous struct?
Anonymous structs are useful when you want to create a struct on the fly.

10. Why do you modularized a struct?
Modularisation makes it easier to reuse the data structure defined in a struct.

Chapter 2: Maps

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Maps by Declaration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Initialization by Make

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Initialization by Literal Type Assignment

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Maps by Assignment

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Assignment with a Key/Value

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Assignment on an Empty Map

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Using Struct in Maps

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Struct Maps by Declaration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Struct Maps by Assignment

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Struct Maps with Array Values

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Maps with Dynamic Types

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Deleting Map Values

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Reading a Non-Present Key from a Map

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Variadic Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Variadic Interface

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Maps with Variadic Interface

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Chapter Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Chapter 3: Arrays and Slices

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Fixed Array

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Fixed-Array Automatic Size Calculation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Fixed-Array Sizes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Fixed-Array Assignment Behaviour

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Sliced Array

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Sliced-Array Assignment Behaviour

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Capacity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Deep Copy

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Append

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Arrays with Variadic Types

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Empty Interface Array Type

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Chapter Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Chapter 4: Array Navigation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

C-style Semantic Form

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Value Semantic Form

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Value Semantic Form with Muted Parameter

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Index Semantic Form for Range

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Value Semantic Form with Pointer Access

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Chapter Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Chapter 5: Package Management

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Sharing Go Packages

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Package Management using Go Modules

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Manual go.mod Generation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Automatic go.mod Generation Through Source-Code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Automatic go.mod Generation Through dep Package Manager

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Refresh Go Modules

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Package Management using Dep

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Chapter Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Chapter 6: Collection Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Predicate Method `all?`

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Predicate Method `any?`

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Collect Enumerable Method

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Cycle Enumerable Method

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Detect Enumerable Method

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Drop Enumerable Method

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Drop While Enumerable Method

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Chapter Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Chapter 7: Interfaces

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Interface as a Self-Documenting API Reference

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Interface as Type contract

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Satisfying Return Values

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Chapter Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Glossary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Acknowledgments

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.

Credits

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rb2go.