FOR

php

DEVELOPERS

Stephen Afam-Osemene

Go for PHP Developers

Learning Go using PHP

Stephen Afam-Osemene

This book is for sale at http://leanpub.com/go-for-php-devs

This version was published on 2019-01-29

Contents

Getting Started

To go through this book you'll need to have both PHP and Go installed. We'll be using PHP 7 and Go 1.11

Here's the official documentation on installing PHP. Link[1]. Here's the official documentation on installing Go. Link[2].

File Extensions

- The file extension `.php` is mostly for convention and convenience. Technically, you can use any file extension for PHP code.
- In Go, files are required to have the `.go` extension to be compiled.

Starting a project

To start a project in PHP, you can just start adding your PHP files.

However in Go, you need the `go.mod` and `go.sum` files to manage your project and its dependencies. >NOTE: This is only available by default in go 1.11 and newer.

go.mod

To create the go.mod file, run the following command from the root of your project.

```
1 go mod init importpath
```

Will will talk more about the `importpath` in the section about importing packages. A good convention is just to use the path to the git repository.

For example, if the remote git repository for my project is https://github.com/stephenafamo/goforphpdevs, I'll run this command to create `go.mod`.

```
1 go mod init github.com/stephenafamo/goforphpdevs
```

[1] http://php.net/manual/en/install.php
[2] https://golang.org/doc/install

go.sum

The `go.sum` file contains the expected cryptographic checksums of the content of specific module versions. Each time a dependency is used, its checksum is added to go.sum if missing or else required to match the existing entry in go.sum.

You don't have to manage this yourself, whenever you add a new dependency and try to run the project, Go will download the dependency and add it to the `go.sum`.

What does the code look like?

In PHP, you tell the interpreter to process PHP code by embedding them inside the tags `<?php` and `?>`.

In Go, every thing inside a `.go` file is expected to be valid Go code.

The first statement in a Go source file must be.

```
1  package name
```

Also, all files in a single directory must have the same package *name*.

The *name* of the package is up to you to decide. However, if you want to run your program, you **must** use *main* as your package name. Like this;

```
1  package main
```

You should use other names when you want to create a library that will be used by other go programs.

We will talk more about that in the section about creating and importing packages. Link.

Running a Program

Now, it may seems like this is moving fast. I mean, we only just installed Go.

However, I believe it is important to see things working as we go along. I promise I won't use anything I have not yet explained.

PHP

To run a PHP program, we would go to the command line and do something like this:

```
1 $ php index.php
```

Go

Go has something similar

```
1 $ go run main.go
```

In Go, you can also compile the entire code as a single executable file using `go build` or `go install`. We'll talk about those much later, but for now we'll mostly use `go run` since it is most similar to how things work in PHP.

How are they different?

Multiple files

PHP

In PHP, you pass a single script to be run. If more files are needed, you can include them in the original file.

GO

In Go, you can pass multiple files within the same directory to be run. A quick way to run all the go files in a directory is to run the command like this;

```
$ go run *.go
```

Order of execution

PHP

When a PHP script is run, the PHP interpreter starts from the beginning of the file and interprets it to the bottom, running commands as it goes from top to bottom.

If there are any included files, it interprets them as though they existed in the file at the point it was included (or required).

This means that an error will occur if a variable (or function) is use *before* it is declared.

Go

Since Go is a compiled language, it will compile all the files before running so it does not matter where a function or variable is declared, as far it is in one of the files.

Instead of starting at the beginning of a file, the `go run` command will look for a function called **main** and start execution from there. It would look like this:

```
package main

func main() {
    // begin code execution
}
```

If the `main()` function is absent, you will see an error like this:

```
$ go run *.go
# command-line-arguments
runtime.main_main·f: relocation target main.main not defined
runtime.main_main·f: undefined: "main.main"
```

Unused variables

In Go, our program will refuse to run if we declare a variable that we do not use anywhere.

To make our project run, we're going to import the `fmt` package and use it to print these variables to the console.

We will treat imports much later in this section. For now you should just try to ignore it.

Variables

Here are the key differences between how variables are handled in Go versus in PHP:

1. Variables must be declared before they are used.
2. All variables must have a **type** which cannot change.

Since PHP is a dynamically typed language, having to think about types may be new to most PHP developers.

In PHP, it is okay to do something like this:

```php
<?php
$a = 1;
$a = "hello";
$a = 0.2;
```

In Go, every variable has a type and will strictly remain that type for the entirety of it's life cycle.

When declaring a variable, you must also state it's type. For example,

```go
package main

import "fmt"

func main() {
    var a int
    a = 1

    fmt.Println(a)
}
```

If you attempt to assign a string to a variable that is an `int` type, the compiler will throw an error. For example:

```go
package main

import "fmt"

func main() {
    var a int
    a = "hello"

    fmt.Println(a)
}
```

```
./main.go:5:9: cannot use "hello" (type string) as type int in assignment
```

Declaring a variable can be done in several ways. Here is a quick summary.

```go
package main

import "fmt"

func main() {
    // "var declares one or more variables".
    var a int
    var b, c string

    // then later
    a = 1
    b = "hello"
    c = "hi"

    // You can do declaration and assignment in the same line
    var d int = 1
    var e, f string = "foo", "bar"

    // If the variable is initialized, the type can be inferred.
    var g = 1 // type inferred as int
    var h = true // type inferred as bool

    // Variables declared without a corresponding initialization are zero-valued.
    // For example, the zero value for an int is 0.
    var i int    // h is 0
    var j string // g is ""

```

```
    fmt.Println(a, b, c, d, e, f, g, h, i, j)
}
```

If we try to use a variable before it is initialized, we will get an error:

```
package main

import "fmt"

func main() {
    x = true
    y = false

    fmt.Println(x, y)
}
```

```
./main.go:4:5: undefined: x
./main.go:5:5: undefined: y
```

Also, if we try to reinitialize a variable, we will get an error.

```
package main

import "fmt"

func main() {
    var x, y bool
    var x, y bool

    fmt.Println(x, y)
}
```

```
./main.go:5:9: x redeclared in this block
    previous declaration at ./main.go:4:9
./main.go:5:9: y redeclared in this block
    previous declaration at ./main.go:4:9
```

Primitive Types

Booleans

In both PHP and Go, the boolean type exists.

A boolean variable can either be `true` or `false`.

```php
<?php
$x = true;
$y = false;
```

```go
package main

import "fmt"

func main() {
    var x, y bool
    x = true
    y = false

    fmt.Println(x, y)
}
```

Strings

Strings exist in both PHP and Go. However, there are some differences.

In PHP, strings can be defined with either single or double quotes. *include note about the differences*

```php
<?php
$x = "Hello world!";
$y = 'Hello world!';
```

In Go, strings **must** use double quotes. We can also use back-ticks “‘” to make string literals. String literals allow us to have multi-line strings

```go
package main

import "fmt"

func main() {
    var x, y string
    x = "this is a single line string"
    y = `this
         is
         a
         multi-line
         string`

    fmt.Println(x, y)
}
```

Integers

In PHP, there is the integer type

```php
<?php
$x = 1;
$y = 2;
```

In Go, there are several integer types:

- int[3]
- int8[4]
- int16[5]
- int32[6]
- int64[7]
- uint[8]
- uint8[9]
- uint16[10]

[3] https://golang.org/pkg/builtin/#int
[4] https://golang.org/pkg/builtin/#int8
[5] https://golang.org/pkg/builtin/#int16
[6] https://golang.org/pkg/builtin/#int32
[7] https://golang.org/pkg/builtin/#int64
[8] https://golang.org/pkg/builtin/#uint
[9] https://golang.org/pkg/builtin/#uint8
[10] https://golang.org/pkg/builtin/#uint16

- uint32[11]
- uint64[12]
- uintptr[13]

```
package main

import "fmt"

func main() {
    var a int = 1
    var b int8 = 2
    var c int16 = 3
    var d int32 = 4
    var e int64 = 5
    var f uint = 6
    var g uint8 = 7
    var h uint16 = 8
    var i uint32 = 9
    var j uint64 = 10
    var k uintptr = 11

    // If no type is specified, the type is inferred as int
    var l = 12 // l has type int

    fmt.Println(a, b, c, d, e, f, g, h, i, j, k, l)
}
```

These are all distinct types and cannot be used interchangeably, however there are ways to do type conversion. We will look at that in this section.

There are also the byte[14] and rune[15] types which are aliases for uint8 and int32 respectively. We'll talk more about this when we discuss type aliases here.

Floating point numbers

In PHP, there is the float type which can be expressed in any of the following ways:

[11] https://golang.org/pkg/builtin/#uint32
[12] https://golang.org/pkg/builtin/#uint64
[13] https://golang.org/pkg/builtin/#uintptr
[14] https://golang.org/pkg/builtin/#byte
[15] https://golang.org/pkg/builtin/#rune

```php
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
```

In Go, there are two float types:

- float32[16]
- float64[17]

```go
package main

import "fmt"

func main() {
    var a float32 = 1.234
    var b float64 = 1.2e3

    // If no type is specified, the type is inferred as float64
    var c = 7E-10 // c has type float64

    fmt.Println(a, b, c)
}
```

Complex numbers

PHP does not directly support complex numbers, there are some libraries which have been created to support it.

On the other hand, Go has two built-in types, complex64[18] and complex128[19].

A complex number has a real and imaginary part. To create a complex number, we use the built-in function `complex()`[20].

The complex built-in function constructs a complex value from two floating-point values. The real and imaginary parts must be of the same size, either `float32` or `float64`, and the return value will be the corresponding complex type (`complex64` for `float32`, `complex128` for `float64`).

[16] https://golang.org/pkg/builtin/#float32
[17] https://golang.org/pkg/builtin/#float64
[18] https://golang.org/pkg/builtin/#complex64
[19] https://golang.org/pkg/builtin/#complex128
[20] https://golang.org/pkg/builtin/#complex

To get the real part of a complex number, we use the built-in function real()[21]. The `real` built-in function returns the real part of a complex number. The return value will be floating point type corresponding to the type of the complex number.

To get the imaginary part of a complex number, we use the built-in function imag()[22]. The `imag` built-in function returns the imaginary part of a complex number. The return value will be floating point type corresponding to the type of the complex number.

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      var a, b float32 = 1.234, 5.678
7      var c, d float64 = 9.012, 3.456
8
9      var e = complex(a, b) // e has type complex64
10     var f = complex(c, d) // f has type complex128
11
12     var g = real(e) // g has type float32
13     var h = imag(f) // h has type float64
14
15     fmt.Println(g, h)
16 }
```

[21] https://golang.org/pkg/builtin/#real
[22] https://golang.org/pkg/builtin/#imag