The Go Programming Language

Marvin Walker  
Daniel Rösch

Inhalt

[2 Einführung in Go 2](#_Toc501485997)

[3 Tooling 2](#_Toc501485998)

[4 Packages 2](#_Toc501485999)

[5 Variablen 2](#_Toc501486000)

[6 Funktionen 2](#_Toc501486001)

[7 Strukturen 3](#_Toc501486002)

[8 Methoden 3](#_Toc501486003)

[9 Interfaces 4](#_Toc501486004)

[10 Map Typen 4](#_Toc501486005)

[11 Slice Typen 4](#_Toc501486006)

[12 Fazit 5](#_Toc501486007)

# Einführung in Go

Go wurde aus Unzufriedenheit über die bestehenden Sprachen zur Softwareentwicklung wie C++ oder Java im Kontext heutiger Computersysteme, insbesondere im Hinblick auf skalierbare Netzwerkdienste, Cluster- und Cloud Computing, entwickelt. Go besitzt im Vergleich zu anderen Programmiersprachen wenig Keywords (Go: 25, C:32, Java, 50). Damit bleibt die Sprache klein und lässt sich so einfach merken. Wichtige Ziele bei der Entwicklung waren unter anderem die Unterstützung von Nebenläufigkeit mit nativen Sprachelementen und die Erleichterung der Softwareentwicklung mit großen Entwicklerteams und großen Codebasen. Go besitzt einen eigenen Garbage Collector, erlaubt die Verwendung von Zeigern, verzichtet jedoch auf Zeigerarithmetik. Go ist eine kompilierte Sprache, bei der Wert auf eine hohe Übersetzungsgeschwindigkeit gelegt wurde. (Auszüge aus Wikipedia)

# Tooling

Mit der Installation von Go werden dem User auch eine diverse Anzahl von Tools mitgeliefert:

* Ein Build Automation Tool
* Ein Coverage Tool
* Ein Tool zum formatieren des Source Codes
* Ein Profiling Tool

Go hat zudem eine ausgezeichnete Dokumentation (<https://golang.org/pkg/>) welcher per Web abrufbar ist. Zudem lässt sich analog zur Dokumentation der Source Code der Standard-Library anzeigen (<https://golang.org/src/> 🡪 *pkg* in der Dokumentations-URL muss durch *src* ersetzt werden). Dies ist insofern wichtig, da der Code der Go Standard Library oft als Referenz für das Go Paradigma genutzt wird.

# Packages

Go lässt sich mittels Packages modularisieren. Da es in Go keine Zugriffsmodifizierer gibt, kommt das Konzept der Exports zum Zuge. Ein Name (kann eine Variable, Funktion, usw. sein) wird exportiert, wenn er mit einem Grossbuchstaben beginnt. Damit ist der Name dann auch ausserhalb des Packages zugänglich.

In Go fehlt jedoch noch ein ausgereiftes Dependency Management. Dieses ist jedoch in Arbeit und soll in absehbarer Zeit mit den Standardtools mitgeliefert werden.

# Variablen

Bei Go handelt es sich um eine stark typisierte Sprache. Trotzdem ist des bei der Variabel Deklaration nicht explizit nötig den Typen anzugeben. Damit fühlt sich Go wie eine dynamisch typisierte Sprache an.

var i int = 3

k := 3

# Funktionen

In Go kann eine Funktion ein oder mehrere Argumente entgegennehmen.

func add(x, y int) (sum int) {

    return x + y

}

Eine Funktion kann in Go eine beliebige Anzahl von Resultaten zurückgeben. Die Rückgabewerte können zudem benannt werden, womit sich eine Funktion sogar noch mit Hilfe des Source Codes klar dokumentieren lässt.

func split(sum int) (x, y int) {

    x = sum \* 4 / 9

    y = sum - x

    return

}

Rückgabewerte einer Funktion können zudem mittels «\_« ignoriert werden.

x, \_ := split(17)

# Strukturen

Ein *struct* ist eine Kollektion von Feldern. Die Syntax ähnelt dabei jener von C. Mit einem *struct* lassen sich also Daten in eine Gruppe zusammenfassen. Da Go keine vollständige OOP Sprache ist, kommt ein *struct* einem Leightweight Objekt am nächsten. Ein Lightweight Objekt, weil es keinen *behavior*  hat sondern nur einen *state.* Mehr dazu im FAQ: <https://golang.org/doc/faq#Is_Go_an_object-oriented_language>

type Vertex struct {

    X int

    Y int

}

func main() {

    v := Vertex{1, 2}

    p := &v

    p.X = 3

    fmt.Println(v)

}

# Methoden

Go hat keine Klassen, jedoch ist es möglich Methoden auf Typen zu definieren. Eine Methode ist jedoch nichts anders, als eine Funktion mit einem *receiver* Argument. Eine Methode kann dabei auch für einen *non-struct* Typen deklariert werden.

func (v Vertex) Abs() float64 {

    return math.Sqrt(v.X\*v.X + v.Y\*v.Y)

}

func main() {

    v := Vertex{3, 4}

    v.Scale(10)

}

# Interfaces

In Go bietet ein Interface die Möglichkeit, Verhalten für einen Typen zu spezifizieren. Ein Interface ist also eine Sammlung von Methoden. Das ein Typ ein Interface erfüllt, muss er mindestens alle Methoden des Interfaces implementieren.

type I interface {

    M()

}

type T struct {

    S string

}

*//* This method means type T implements the interface I,

*//* but we don't need to explicitly declare that it does so.

func (t T) M() {

    fmt.Println(t.S)

}

func main() {

    var i I = T{"hello"}

    i.M()

}

# Map Typen

Go bietet zudem als built-in Typen eine Map an. Dabei wird wie gewohnt ein Wert zu einem bestimmten Schlüssel zugeordnet.

*//* 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)

# Slice Typen

Der Slice Datentyp ist einem Array sehr ähnlich. Ein Slice bietet jedoch noch zusätzliche Funktionen, welche den Umgang mit Datensequenzen erleichtern. Im Gegensatz zum Array hat ein Slice keine fixe Länge.

*//* Slice Literal

letters := []string{"a", "b", "c", "d"}

*//* make-Funktion

var s []byte

s = make([]byte, 5)

*//* s == []byte{0, 0, 0, 0, 0}

# Fazit

**Marvin Walker:**  
Go wurde in erster Linie von und für Google entwickelt und nicht für die breite Öffentlichkeit. Dass Go trotz diesem Aspekt heute zu einer verbreiteten Programmiersprache gehört, zeugt von grosser Überzeugungskraft der Sprache. Meiner Meinung nach eignet sich Go optimal als Einsteiger Programmiersprache. Die gängigsten Programmierkonstruktionen wie z.B. einen Webserver können mit minimalstem Code Aufwand erzeugt werden. Aber nicht nur Anfänger können sich an der Sprache Erfreuen, denn auch die Experten von Google setzen für Ihre komplexen Algorithmen auf die Sprache Go. Dies zeigt, dass Go das Zeug hat für einen Karrieren langen treuen Begleiter.

**Daniel Rösch:**  
Go wirkt simpel und aufgeräumt. Wenn man mit der Syntax von C vertraut ist, findet man sich sehr schnell mit der Sprache zurecht. Nützlich sind auch die in der Sprache integrierten Möglichkeiten der Nebenläufigkeit. Auch das Tooling von Go mag zu überzeugen, womit man auf externe Tools fast gänzlich verzichten kann. Mit Google als Entwickler von Go, kann man sich sicher sein, dass man auf eine langjährige und kontinuierliche Weiterentwicklung der Sprache zählen darf.