Programmieren lernen mit Go (golang)

Version 1.0.1, Juni 2022, Jens Schendel

Inhalt

Lektion 1 – Willkommensgruß, Einladung zum Lernen von Go und Hinweise zu Kurs und Übungen

Willkommensgruß

Lektion 2 – Warum ausgerechnet Go? Historie, Einordnung und Typisierung

Wer hat’s erfunden?

- Google (zu Wikipedia)

- Rob Pike

    Unix, UTF-8

- Robert Griesemer

    Hat studiert beim Erfinder von Pascal

- Ken Thompson

    federführend bei der Implementierung von Unix

    hat die B Programmiersprache und damit den Vorläufer von C erfunden und beteiligt, C ins Leben zu rufen.

Warum jetzt genau nochmal Go?

- Hocheffizientes Compilieren
- Go erstellt kompilierte Programme
- Es gibt einen „Garbage Collector“ (GC)
- Es gibt kein virtuelle Maschine, in der Code ausgeführt wird, keinen Emulator und keinen Interpreter
- Schnelle Ausführungzeiten
- Einfacheit in der Anwendung, „Ease of programming“

Zusammengefasst **drei Hauptmerkmale**, die Go so erfolgreich machen:

1. Compiliert einfach und sehr schnell – auch große Projekte compilieren in Sekunden und Minuten und nicht in Stunden.
2. Effizient Ausführung mit sehr hoher Ausführungsgeschwindigkeit
3. Ease of Programming – Programmieren soll mit Leichtigkeit vonstatten gehen, keine Krämpfe in Gehirnwindungen.

Wofür man Go gut nutzen kann

- Alles, was Google macht / alle Intenet Dienste, die höchsten Ansprüchen genügen müssen und hochskalierbar sein müssen.
- networking
- http/https, tcp, udp
- Nebenläufigkeit / Parallelprogrammierung
- bedingt Systemprogrammierung
- Automation, command-line tools
- Crypto
- Bildverarbeitung

Erstellungsprinzipien

- aussagekräftig, verständlich, anspruchsvoll
- sauber, klar, leicht zu lesen

Unternehmen, die Go einsetzen

> Unter anderem bei Google, YouTube, Netflix, Google Confidential, Docker, Kubernetes, InfluxDB, Twitter, Apple, Cloudflare, DropBox und andere, weitere Beispiele im Detail

Trivia:

Der Erfinder von Node.js hat Node aufgegeben und gibt stattdessen Go den Vorzug

Go programmierer sind zur Zeit die bestbezahlten Programmierer in den USA – Platz 5 weltweit

Lektion 3 – Lernhinweise zu diesem Kurs

Lektion 4 – Übersicht über Vorgehensweise

Lektion 5 – Kursübersicht als PDF und auf Github

https://github.com/Educational-Coding-Examples-Exercises/go-collection/blob/main/README.md

Lektion 6 – Informationsquellen im Web zu Go von der Herstellern/Enwicklern

Lektion 7 – Terminal/Konsolen/Shell

Eine kurze Klärung der Terminologie.

Lektion 8 – Bash für Windows

Unter MS Windows werden Sie keine unixoide Eingabeaufforderung zwingend nutzen müssen, aber

Sie können dies tun:

https://www.cygwin.com/

oder Windows Subsystem for Linux

https://docs.microsoft.com/de-de/windows/wsl/install

Lektion 9 – Einfache Einführung in die bash

https://www.ernstlx.com/linux90bash.html und falls Sie grundlegende Kenntnisse über Linux benötigen, empfehle ich Ihnen diesen kostenlosen (Englisch) Videokurs von Shawn Powers – in mehr als 5 Stunden bringt Ihnen der Dozent in unterhaltsamer und leicht verständlicher Weise die Linux-Grundlagen bei.

Lektion 10 – Einfache Einführung in die Eingabeaufforderung

https://www.thomas-krenn.com/de/wiki/Cmd-Befehle_unter_Windows

Lektion 11 – Installation von Go auf Linux, MAC und (Windows)-PC

Kurze Einweisung in die Installation von go.

Für das Zielsystem passenden Installer oder Paket herunterladen, wenn nötig entpacken, installieren, bzw, durch einen Paketmanager installieren lassen.

Bitte auf die richtige Architektur (386, AMD64, ARM in 32-bit, 64 bit) achten. Wer will, mag ggfls. auch die Prüfsummen mit denen auf der Website angegebenen vergleichen, um Manipulationen auszuschließen. Eventuell ist der Pfad zu Go dem Standardpfad des Benutzers (oder aller Benutzer) hinzuzufügen.

Link im Web: https://go.dev/dl/

Lektion 12 – Umgebungsvariablen

Lektion 13 – Programmerstellung, Kompilierung, Ausführung

Quellcode schreiben, kompilieren, ausführen.

Lektion 14 – Native Go-Befehle

`go env` – zeigt uns die durch Go gesetzten Umgebunsgvariablen an

`go fmt code.go` – formatiert, wenn keine groben sysntaktischen Fehler vorliegen, den Go Code `code.go`

`go run code.go` – compiliert den Quellecode `code.go` und bringt das Ergebnis zur Ausführung.

Lektion 16 – Off-Topic: Git installieren und github benutzen

Was ist git? https://de.wikipedia.org/wiki/Git

Git installieren: https://git-scm.com/downloads

github.com (gitlab.com): https://www.github.com

Vorschlag TL;DR;

Installieren Sie git auf Ihrem lokalen Rechner.

Erstellen Sie einen kostenlosen Account und ein Repository auf github (mit .gitignore Datei für Go)

Erstellen Sie einen Access Token (mit Geltungbereich (Scope „Repo“)) oder machen Sie sich mit der SSH-Authentifizierung vertraut.

Auf Ihrem lokalen Rechner wechseln Sie in ein Verzeichnis, in dem Sie Ihre Projekte verwalten wollen. Dort geben Sie ein:

```
git clone https://github.com/accountname/repositoryname.git
```

bei SSH-Zertifizierung bei github.com lautet die Eingabe:

```
git clone git@github.com:accountname/repositoryname.git
```

Sie werden nach Accountname und Access Token (als Passwort) gefragt.

Sie haben eine lokale Kopie des Repository.

Ändern Sie etwas an Ihrem Repository lokal, können Sie die Änderungen (wenn Sie im Verwzeichnis sind) hochladen mit:

```
git add .
git commit -m „eine aussagekräftige Nachricht“
git push origin main
```

Wollen Sie bereits (woanders oder von anderen) ausgeführte Änderungen von github lokal updaten können Sie das mit:

```
git pull origin main
```

Sich überschneidene Änderungen bedürfen besonderer Aufmerksamkeit, schauen Sie dazu auf github.com, wie im Einzelfall zu verfahren ist.

Lektion 17 – Idee für eine IDE: Der Go Spielplatz

Playground

run

Führt Go Code im Broswer aus, indem es den Code zum Webserver des Playground übertragt, übersetrzt, dort ausführt und nur die Ausgabe zurück überträgt.

fmt

Warum braucht es „idiomatic go”?

Portability und Teamarbeit an derselben Codebase wird ungleich vereinfacht, wenn man sich an die gleichen Regeln hält. „Format“ oder auch oft „famp“ gesprochen, sorgt dafür, dass diese idiomatischen Regeln eingehalten werden

- „Idioms“ selbst sind Sprachmuster gebräuchliche Sparchmuster, sowohl in gesprochener Sprache als auch in Programmiersprachen

Beispiel: „Raining cats and dogs“ würde in Deutsch eher ein „Es schüttet wie aus Kübeln“ enstprechen. Verschiedene Idioms in verschiedenen Sprachen.

- Wenn von “idiomatic Go” gesprochen wird ist damit gemeint, Go Code so zu schreiben, wie es in der Go Community üblich ist und von Google angedacht ist.
- Die Funktion „Format“ im Playground, ebenso wie ein „go fmt gofile.go“ auf der Konsole stellen sicher, dass Go Code den idiomatischen Ansprüchen genügt (was vermutlich auch nicht zuletzt dafür sorgt, dass der Compiler den Code schnell verarbeiten kann)

share

So kann man sicherstellen, dass auf Foren wie https://forum.golangbridge.org/ Code leicht ausgetauscht werden kann und jeder sofort gleiche Formatierungen einhält.

Ideal für diesen Kurs, um kleine Beispiele zu teilen und in dieser Kursübersicht zur Verfügung zu stellen.

Beispiel: https://go.dev/play/p/c4Ly4WPBZ9d

Lektion 18 – Hello World, Kontrollstrukturen

Kontrollstrukturen (control flow)

https://de.wikipedia.org/wiki/Kontrollstruktur

Sequence

Go-Code wird „der Reihe nach von oben nach unten und von links nach rechts“ gelesen, interpretiert und ausgeführt.

Schleifen

Es gibt in Go keine while oder do...while-Schleifen – zumindest nicht als Keywords. Alle Schleifen werden als for-Schleifen realisiert.

Beispiel:

:= deklariert und initialisiert eine Variable mit einem Startwert.

```
for Variable := startwert; Bedingung bis zu der die Schleife laufen soll; Änderungsanweisung der  Variablen {Codeblock zur Ausführung, in dem die Variable und deren Wert zur Verfügung steht}
```

Bedingungen

Bedingungen (conditionals) prüfen, ob eine oder mehrere Bedingungen zutreffen und führen ggfls. Codeblock aus.

Beispiel:

```
if Bedingung {Codeblock zur Ausführung, wenn Bedingung zutrifft}
```

Codebeispiel Kontrollstrukturen: https://go.dev/play/p/tciECuRpbOu

Lektion 19 – Exkurs Packages, variatische Parameter

Einführung in Packages

Variatische Parameter (variadic parameters)

- Die Notation “...<some type>” ist nötig, um variatische Parameter anzugeben
  der Dateityp “interface{}” ist ein sogenanntes leeres Interface, jeder Wert der dort folgt, ist ebenfalls vom Typ “interface{}”
- Das bedeutet soviel wie, dass “...interface{}” erlaubt, beliebige viele Werte und Argumente von jedwedem Typ zu übergeben

Rückgabewerte ignorieren/verwerfen

- Man nutzt “_” Unterstrich und Rückgabewerte zu verwerfen

Warum darf man in Go keine ungenutzten Variablen haben?

- Codeverschmutzung (code pollution)
- Der Compiler erlaubt es nicht

Notation zur Nutzung von Funktionen aus importierten Packages in Go

```
package.Identifier
```

Beispiel:
```
fmt.Println()
```

- Bedeutung in etwa “von Package fmt nutze die funktion Println()”
- Als identifier dient der Name einer Variablen, Konstanten oder Funktion.

Packages

- Packages enthalten vorgeschriebenen Code, den man importieren und benutzen kann
- ähnlich includes von Header-Dateien in C.

Codebeispiel Packages, variatische Parameter: https://go.dev/play/p/oqTmqB-WRhk

Lektion 20 - Terminologie und der Short Declaration Operator

Terminologie

In Go unterscheiden wir:

Keywords

Alle Zeichenfolgen, die in Go bereits zur Nutzung vordefiniert sind.

- manchmal auf „reservierte Ausdrücke“ o.ä. genannt
- ein Keyword kann ausschließlich für seinen von den Herstellern von Go bestimmten Zweck benutzt werden

Operatoren

- in “2 + 2” ist “+” der Operator
- Ein Operator ist ein Zeichen, das eine Operation darstellt, wie das “+” einen arithmetischen Operator für das erstellen einer Summe darstellt

Operanden

- in “2 + 2” sind die “2”en die Operanden.

Statements

Beim Programmieren ist ein Statement die kleines Einheit, der eine Anweisung für ein Programm enthalten kann, um eine Aktion auszuführen. Ein Programm entsteht durch die Aneinandereihung von Statements, die als Sequenz ausgeführt werden

Expressions (Ausdrücke)

In der Programmierung ist ein Ausdruck eine Aneinandereihung von Werten, Konstanten, Variablen, Operantoren und Funktionen, die von der Programmiersprache interpretiert und schließlich ausgeführt werden, um einen Wert daraus zu erhalten.

So ist 2 + 3 ein Ausdruck, aus dem sich der Wert 5 ergibt.

Beispiel Short Declaration Operator:

```
var variablenname int
variablenname = wert
```

abgekürzt als

`variablenname := wert` (das impliziert in diesem Fall den Typ Integer)

Beispiel: https://go.dev/play/p/ktnV5CcoGQY

Lektion 21 - Keyword var - das darf ja wohl nicht var sein!

Kurze Wiederholung

Beispiel: https://go.dev/play/p/vYtleP18dJs

```
var y = 23
```

Deklaration der Variable "y"

Wertzuweisung: Wert ist 23

Implizierte Typzuweisung und damit Initialisierung

Deklaration UND Wertzuweisung = Initialisierung

```
var z int
```

Deklaration der Variable "z"

Typzuweisung: Identifier "z" ist Typ int

Implizierte Wertezuweisung und Initialisierung durch automatische Zuweisung eines "Null"-Wertes (Zero Value, in manchen Fällen nil) d. h. z. B.: false für booleans, 0 für integers, 0.0 für floats, "" für strings und nil für pointers, functions, interfaces, slices, channels und maps.

Lektion 22 - Typisierung - genau mein Type

Es gibt da den Spruch „Go suffers no fools.” Frei übersetzt: Go leidet keinen Mangel an Narren. Man könnte Fools vielleicht auch mit Dummköpfen übersetzen, aber Narren scheint mir weniger beleidigend.

Wenn man eine **Variable** als von einem bestimmten **Type deklariert,** kann diese Variable auch nur **Werte/Values** eines bestimmten Typs halten.

Beispiel: https://go.dev/play/p/clyuLTvww7k

```
var z int = 23
```

außerhalb einer Funktion deklariert hat z „package scope“ also paketweite Geltungsbereich.

Primitive Datentypen (Primitive Types)

In der Informatik ist ein primitiver/elementarer Datentyp einer der folgenden Typen:

- ein Basistyp ist ein Datentyp, der von einer Programmiersprache als Grundbaustein bereitgestellt wird. Die meisten Sprachen erlauben es, kompliziertere zusammengesetzte Typen ausgehend von Basistypen zu konstruieren.
- ein eingebauter Typ ist ein Datentyp, für den die Programmiersprache eine eingebaute Unterstützung bietet. In den meisten Programmiersprachen sind alle Grunddatentypen eingebaut. (int, float, char string usw)

Darüber hinaus viele Sprachen auch eine Reihe von zusammengesetzten Datentypen. Die Meinungen gehen auseinander, ob ob ein eingebauter Typ, der nicht grundlegend ist, auch als "primitiv" betrachtet werden sollte.

Die Seite https://en.wikipedia.org/wiki/Primitive_data_type gibt's so nicht in Deutsch auf Wikipedia, aber https://de.wikipedia.org/wiki/Datentyp#Elementare_Datentypen erklärt ganz gut, was gemeint ist.

Zusammengesetze Datentypen (composite data types)

In der Informatik ist ein zusammengesetzter Datentyp oder ein zusammengesetzter Datentyp jeder Datentyp, der in einem Programm aus den primitiven Datentypen einer Programmiersprache und/oder aus anderen zusammengesetzten Datentypen konstruiert werden kann. Er wird manchmal auch als Struktur- oder aggregierter Datentyp bezeichnet, obwohl sich der letztere Begriff auch auf Arrays, Listen, etc. beziehen kann. Der Vorgang der Konstruktion eines zusammengesetzten Typs wird in English oft als „Composition" bezeichnet.

Auf der deutschen Wikipediaseite https://de.wikipedia.org/wiki/Datentyp#Zusammengesetzte_Datentypen werden eben auch Strings und Arrays als zusammengesetzte Datentypen bezeichnet.

Beispiel: https://go.dev/play/p/clhJGk3a0g6

Lektion 23 - Nullnummern - der Zero-Wert, also `nil`

Notwendigkeit eines Nullwertes.

Es geht bei einer stark typisierenden Sprache wie Go darum, auch beim Deklarieren einer Variable einen Werte zuzuweisen, damit man nicht – wie zum Beipsiel in C – plötzlich mit einem Fantasiewert dasteht, nachdem man eine Variable deklariert, aber ihr noch nicht explizit einen Wert zugewiesen hat.

Ease of programming – Bei der Programmieren in Go soll ja Leichtigkeit mitschwingen.

Verschiedene Typen erhalten verschiedene „Nullwerte"

- false für booleans

- 0 für integers

- 0.0 für floats

- "" für strings

- `nil` für
    - Pointer (Zeiger)
    - Funktionen
    - Interfaces
    - Slices
    - Channels
    - Maps

https://de.wikipedia.org/wiki/Nullwert

Eine allgemein als „Best Practice“ empfohlene Arbeistweise ist, den Short Declaration Operator so häufig wie möglich zum Einsatz zu bringen, aber var zu nutzen für

- Nullwerte (zero value)
- Paketweiten Gültigkeitsbereich (package scope)

Beispiele: https://go.dev/play/p/srWjYSVhQnh

Lektion 24 - Das Package fmt - bringt unseren Code so richtig in Form

Das Package fmt bringt grundsätzliche Funktionen zur Ein- und Ausgabe in Go mit.

Übersicht: https://pkg.go.dev/fmt#pkg-overview

Beispiel für „verbs“ wie „%v“ innerhalb von Strings: https://go.dev/play/p/4kcVtZXMjvU

Beispiel verschiedene „rune riterals“ / „escaped“ Zeichen wie \n or \t:
https://go.dev/ref/spec#Rune_literals

Formatierte Ausgabe

Unterschiede der verschiedenen ein- und ausgabeoerientierten Funktionen:

https://pkg.go.dev/fmt#Print

https://pkg.go.dev/fmt#SPrint

https://pkg.go.dev/fmt#Fprint

Beispiel: https://go.dev/play/p/JypKbYzKE9R

Gruppe 1: Allgemeine Ausgabe auf stdout (Standardausgabe)

- func Print(a ...interface{}) (n int, err error)
- func Printf(format string, a ...interface{}) (n int, err error)
- func Println(a ...interface{}) (n int, err error)t

Gruppe 1: Allgemeine Ausgabe in einen String (string, daher vorangestelltes „s“), was in unserem Falle aber auch eine Variable vom Typ String sein kann!

- func Sprint(a ...interface{}) string
- func Sprintf(format string, a ...interface{}) string
- func Sprintln(a ...interface{}) string

**Gruppe 1: Allgemeine Ausgabe in eine Datei (file, daher vorangestelltes „f“),
was in unserem Falle aber auch eine Antwort eines Servers o.ä sein kann!**

- func ww(w io.Writer, a ...interface{}) (n int, err error)
- func Fprintf(w io.Writer, format string, a ...interface{}) (n int, err error)
- func Fprintln(w io.Writer, a ...interface{}) (n int, err error)

Lektion 25 - Do it yourself: Type im Eigenbau

In Go können wir natürlich auch unsere eigenen (auch zusammengesetzten) Typen erstellen.

Beispiel: https://go.dev/play/p/9WxlCBHxTu-

Lektion 26 - Typveränderung: Conversion ist nicht Casting!

Go hat seine eigene Sprache, um über sich und von sich selbst zu sprechen. Alte Begrifflichkeiten wurden über Bord geworfen, weil sie mit Altlasten beladen daherkommen. In Go wurde Programmierung neu erdacht und daher werden auch neue Bezeichnungen benutzt, um über einige Konzepte zu sprechen und deren Besonderheiten gerecht zu werden.

So wird in Go nicht mehr von Objekten gesprochen, sondern von der Erstellung von Typen und Werten eines bestimmten Typs (value of type). Natürlich spiegelt sich viel von Object Orientated Programming (OOP) auch in Go wider, aber die Begriffe sollte man vermeiden, weil im Zweifelsfall sich Konzepte und Anwendung im Detail von anderen Programmiersprachen unterschieden!

So sprechen wir bei Go auch nicht von „Casting“, sondern von „Conversion“ (und „Assertion").

Beispiel: https://go.dev/play/p/LDlF_ZbgH4P

Lektion 27 – Hinweise zu den Übungen

Einfach nur ein paar Hinweise zur Motivation!

Takeaway: Machen!

Lektion 28 – Übung 1

1. Erstellen Sie mit dem Short Declaration Operator die Variablen mit den Identifiern „x”, „y” und „z” und weisen Sie ihnen folgenden Werte zu:

a) 23

b) „Papa Schlumpf”

c) true

2. Geben Sie die Werte der Variablen aus mit

a) einem einzelnen „print“-Statement

b) mehreren einzelnen „print“-Statement

Lektion 29 – Übung 1 Beispiellösung

https://go.dev/play/p/j8JzP4yH6Lm

Lektion 30 – Übung 2

1. Erstellen Sie mit dem Keyword var global die Variablen mit den Identifiern „x”, „y” und „z” und deklarieren sie deren Typen als:

a) int

b) string

c) bool

2. Geben Sie die Werte der Variablen in der Funktion main() aus.

Zusatzfrage: Wie nennt man diese vom Compiler zugewiesenen Werte?

Lektion 31 – Übung 2 Beispiellösung

https://go.dev/play/p/hHYDYExC6-x

Lektion 32 – Übung 3

Auf Grundlage ihres Codebeispieles aus der vorherigen Übung

1. weisen Sie den drei Variablen auf der Gültigkeitsbereichebene des ganzen Paketes die Werte

a) 23

b) „Schlumpfine”

c) true

zu und in Funktion main()

2. Benutzen Sie die Funktion main Sprintf

a) um alle drei Werte, einer eigenen Variablen mit dem Identifier „s“ zuzuweisen, die Sie mittels Short Declaration Operator erstellen,

b) und geben Sie den in „s“ gespeicherten Wert aus.

Lektion 33 – Übung 3 Beispiellösung

https://go.dev/play/p/1rhjXq-z5iJ

Lektion 34 – Übung 4

FYI: Einfache Erklärung der Terminology “underlying type”

https://go.dev/ref/spec#Underlying_types

Für diese Übung

1. Erstellen Sie Ihren eigenen Variablentyp basierend auf dem Typ int.

2. Erstellen Sie die Variable „x“ mit dem von Ihnen erstellen Typ mittels var

3. In der Funktion main()

   a) Geben Sie den Wert von „x” aus

   b) Geben Sie den Typ von „x” aus

   c) Weisen Sie „x” den Wert 23 mit dem einfachen Zuweisungsoperator zu

   d) Geben Sie nochmal den Wert von „x” aus

Lektion 35 – Übung 4 Beispiellösung

https://go.dev/play/p/DhvNphCwYGk

Lektion 36 – Übung 5

Auf Grundlage ihres Codebeispieles aus der vorherigen Übung

1. Erstellen mit var eine Variable mit dem Identifier „y“ auf der Gültigkeitsbereichebene des ganzen Paketes und weisen Sie ihr den „underlying type“ ihres eigenen erstellen Types zu (also int).

2. In Funktion main()

Weisen Sie „y“ den Wert von „x“ zu und nutzen Sie „conversion“, also die Umwandlung des Wertes der Variablen „x“ in den underlying type int.

3. Geben Sie den Wert von „y“ aus

4. Geben Sie den Typ von „y“ aus

Lektion 37 – Übung 5 Beispiellösung

https://go.dev/play/p/Nsu3XX441Vf

Quiz 1

Lektion 38 – Übung 6 Quiz Lösung

Lektion 39 – Bool Type: Sein oder nicht Sein!

https://de.wikipedia.org/wiki/George_Boole

https://de.wikipedia.org/wiki/Boolesche_Algebra

Beispiele:

https://go.dev/play/p/AHr7iSOcnhK

https://go.dev/play/p/znA51euWOSk

Lektion 40 – Exkurs zu Binärzahlen und wie Computer arbeiten

Computer rechnen intern ausschließlich mit Binärzahlen, also im Zahlensystem mit der Basis 2.

Nullen und Einsen werden im Binärsystem wie Schalter für Lampen betrachtet. Jeder dieser Schalter nennt sich ein „Bit“ (eine zusammengesetzter Begriff aus „Binary Digit“. Diese Bits bedürfen der Interpretation durch uns. Allgemein nennt man acht solcher Bits ein Byte.

10101010 ist ein Beispiel für ein Byte. Ein Byte kann alle Werte von 0 bis 255 also insgesamt 256 Werte darstellen.

Werte für jede Stelle (von rechts nach links) werden mit 0 und 1 beschrieben und jede höhere Stelle stellt eine um 1 höhere 2er Potenz dar.

Beispiel:

23 im Dezimalsystem (Zahlensystem mit der Basis 10) entspricht der Binärzahl:

0	0	0	1	0	1	1	1	(16 + 4 + 2 + 1 = 23)
128	64	32	16	8	4	2	1	

https://de.wikipedia.org/wiki/Bin%C3%A4rcode

Eine weitere Interpretation kann bereits im Computer stattfinden, aber muss nach einem einheitlichen zuvor abgesprochenen System standartisiert werden. Ein Beispiel dafür ist das ASCII-System, dass Zahlenwerten Buchstaben und Zeichen zuordnet.

https://de.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange#ASCII-Tabelle

ASCII nutzt 7 Bit zur Beschreibung.

UTF-8 nutzt 32 Bits und kann damit über eine Millionen verschiedener Zeichen abbilden, theoretisch sogar mehr als 2 Millionen.

https://de.wikipedia.org/wiki/UTF-8

Die grundsätzliche Funktionsweise von Computern beruht auf der Fähigkeit, sehr schnell Rechnenoperationen mit Binärzahlen ausführen zu können. Anfangs mittels (ganz änfänglich noch mechanischen) Schaltungen und Lampen, später mittel Kondensatoren, dann in integrierten Schaltungen und schließlich auf universell programierbaren CPUs (Central Processing Unit), die wir heute kennen. Die Anzahl von Schaltungen in Form von Transistoren und damit die Rechenleistung scheint sich in den letzten Jahrzehnten in regelmäßigen Intervallen verdoppelt zu haben. https://de.wikipedia.org/wiki/Mooresches_Gesetz

Lektion 41 – Numerische Typen

Numerische Typen beschreiben Zahlen.

https://go.dev/ref/spec#Numeric_types

Die Typen int, float und complex stehen für die Menge der Ganzzahl-, Gleitkomma- bzw. komplexen Zahlen. Sie werden zusammenfassend als numerische Typen bezeichnet.

Integers beschreiben ganze Zahlen.

Floats beschreiben Zahlen mit Nachkommastellen.

Complex beschrieben komplexe Zahlen (hier vernachlässigt).

Die Größe eines Typs kann architektur-unabhängig angegeben werden. Die verschiedenen „Untertypen“ sind nicht zueinander kompatibel. Ein int32 und ein int sind daher nicht untereinander austauschbar, auch wenn ihre Größe auf vielen Architekturen dieselbe ist! Stichwort auch hier: (strenge) statische Typisierung!

Integers unterscheiden sich nochmal in mit und ohne Vorzeichen. Das linke Bit wird als Indikator für ein Vorzeichen genutzt, dadurch „halbiert“ sich mögliche darstellbare Größe von „unsigned int“ für Integers mit Vorzeichen.

byte Alias für uint8

rune Alias für int32

Daumenregel: Einfach int und float64 nutzen. Was gut genug für den Compiler ist, ist auch gut genug für uns. https://go.dev/play/p/2QlV3pguOEA Aber wir können den Typ genau angeben, wenn wir wollen: https://go.dev/play/p/Y8sag2jIedM

Wer Speicherplatz sparen will (oder muss), kann das also einfach tun. Beispiel wären Funktionen, deren einfache Zähler ohnehin sich immer im unteren dreistelligen Bereich bewegen, oder Packages, die massive Parallel ablaufen und ansonsten zu viel nie genutzten Speicher belegen.

Package `runtime` bietet `GOOS` und `GOARCH`

https://go.dev/play/p/nJCcrYrxfDK

Lektion 42 – String ist ein Typ

Strings sind in Go ein eigener Datentyp.

TL;DR;

1. Strings sind in Go ein eigener Datentyp.

2. Die Werte in Strings sind unabänderbar (immutable, read only).

3. Stringwerte sind „Slices of Byte (uint8)“.

4. Strings können leer sein.

Beispiele im Video:

https://go.dev/play/p/J0E71TdQ2Wc

https://go.dev/play/p/6kOq6hgWkuI

https://go.dev/play/p/C8yrp2eAufi

Gut zu Wissen:

Die „Slices of Byte“ beruhen auf einer Datenstruktur „Pointer auf den Anfang des Slices“ und „Länge des slices in Byte“ d.h. die Werte bedürfen auch einer anschließenden Interpretation. Das hat schon viel Ähnlichkeit mit bekannten Konzepten wie Arrays von Characters, ist aber weniger eingeschränkt.

Erklärung wie Zeichen überhaupt ausgegeben werden, wenn doch in Go nur Daten vom Typ byte aneinandergereicht werden: https://golangbyexample.com/character-in-go/

Erklärung und weiterführende Informationen von Rob Pike himself, die hilft UFT-8 in Go zu verstehen und Daten vom String nicht nur als Aneinanderreihung von Zeichen zu sehen:

https://go.dev/blog/strings

Wer mag kann auch schon mal „vorlernen“ und sich ansehen, was ein Slice ist und warum das kein array ist, sondern eine Struktur in Go, die sich von der Einschränkungen eines arrays befreit hat und viel mehr Möglichkeiten bietet: https://go.dev/blog/slices

Lektion 43 – Zahlensysteme: 2, 8, 10, 16 - binär, oktal, dezimal oder hexadezimal

Wir lernen Zahlensystem (nochmal) kennen.

Dezimalsystem (zur Basis 10)

0 1 2 3 4 5 6 7 8 9 = Zehn Ziffern

10Ter	1000er	100er	10er	Einer		
10^4	10^3	10^2	10^1	10^0 (1)		
1	2	3	4	5	=	12345

Binärsystem (zur Basis 2)

0 und 1 = Zwei Ziffern

16er	8er	4er	2er	Einer		
2^4	2^3	2^2	2^1	2^0		
1	0	1	1	1	=	23

Binärzahl 32 Bit mit allen Stellen ausgeschrieben: 00000000000000000000000000010111 entspricht 23 im Dezinmalsystem.

Hexadezimal (zur Basis 16)

0 1 2 3 4 5 6 7 8 9 A B C D E F = Sechszehn Ziffern

65536er	4096er	256er	16er	Einer		
16^4	16^3	16^2	16^1	16^0		
0	0	0	1	7	=	23
0	0	0	1	1	=	17
0	0	0	7	B	=	123

Beispiel Ausgabe in Go bis hier: https://go.dev/play/p/TB-f0IsqFV6

Oktal (zur Basis 8)

0 1 2 3 4 5 6 7 = Acht Ziffern

4096er	512er	64er	8er	Einer
8^4	8^3	8^2	8^1	8^0

Einfache Umrechnungstabelle online:
https://www.elektronik-kompendium.de/sites/dig/0710081.htm

Zahlensysteme: https://de.wikipedia.org/wiki/Zahlensystem

Letzte Beispielausgabe in Go: https://go.dev/play/p/fAayLqZgN_r

Lektion 44 – Konstanten - die Konstanten im Leben und in Go

> **„Die einzige Konstante im Leben ist die Veränderung“** – Heraklit

Konstanten, Beispiele für Deklarierung: https://go.dev/play/p/dyJu_BBGtqs

Lektion 45 – Iota

Iota ist ein vordefinierter Identifier, der während der Deklaration von Konstanten benutzt werden kann, um eine bei jeder Zuweisung im 1 erhöhte Ganzzahl nutzen zu können.

Iota Beispiele: https://go.dev/play/p/VgNTaj-U4tj

Lektion 46 – Bit shifting: Verschiebebahnhof!

Bits kann man in Go mit einfachen Mittel verschieben und so Werte von Variablen und Konstanten manipulieren!

Einfaches Beispiel bit shifting: https://go.dev/play/p/xMvj6ako5HV

Komplexes Beispiel bit shifting (mit Konstanten und Iota): https://go.dev/play/p/szGbnPhwwtB

Beispiel anderer bit-manipulierender Operatoren: https://go.dev/play/p/fhPRztZQHu0

Beitrag Medium zu Bit Operatoren:
https://medium.com/learning-the-go-programming-language/bit-hacking-with-go-e0acee258827

Lektion 47 – Weitere Hinweise

Takeaway: **Machen, auch wenn einfach oder zu schwer!**

Lektion 48 – Übung 1

Schreiben Sie ein kurzes Programm, das einer Variablen den Typ uint32 zuweist. Weisen Sie dieser Variablen einen Wert aus ihrem Wertebereich zu.

Geben Sie den Wert als Dezimalzahl, als Binärzahl und in hexadezimaler Schreibweise (mit vorangestelltem „0x“) aus.

Lektion 49 – Übung 1 Beispiellösung

https://go.dev/play/p/uBTJFYwDDqB

Lektion 50 – Übung 2

Nutzen Sie die nachfolgenden Operatoren und erstellen Sie je einen Ausdruck damit, deren Werte (Auswertung) Sie jeweils einer Variablen zuweisen (mittels Short Declaration Operator).

1. == in Variable a
2. <= in Variable b
3. >= in Variable c
4. != in Variable d
5. < in Variable e
6. > in Variable f

Geben Sie die Werte der Variablen a bis f mit nur einem Statement untereinander aus.

Lektion 51 – Übung 2 Beispiellösung

https://go.dev/play/p/uqqeb3vFR9t

Lektion 52 – Übung 3

Erstellen Sie sowohl „typed“ als auch „untyped“ Konstanten.

Geben Sie in einem Statement die zugewiesenen Werte und zugewiesenen/angenommenen Typen aus.

Lektion 53 – Übung 3 Beispiellösung

https://go.dev/play/p/squOAZP2LED

Lektion 54 – Übung 4

Schreiben Sie ein Programm, das

- einer Variable den Typ int und den Wert 23232 zuweist.
- Geben Sie diesen Wert nebeneinander als binär, dezimal und hexadezimal aus.
- Weisen Sie das um 1 nach links verschobene Bitmuster diesen Wertes einer neuen Variable zu.
- Geben Sie den Wert dieser Variablen nebeneinander als binär, dezimal und hexadezimal aus.

Tipp: „Verb“ für die Ausgabe in Binärschreibweise ist „%b“ und Breite des Ausdrucks auf 32 Stellen erweistert und (nach links) mit Nullen aufgefüllt: „%032b“.

Lektion 55 – Übung 4 Beispiellösung

https://go.dev/play/p/a6E6ByCRJfo

Lektion 56 – Übung 5

Erstellen Sie mit dem Short Declaration Operator eine Variable vom Typ `string` und weisen Sie ihr als Wert mittels eines „raw string literal“ zu. Der Wert sollte einen Zeilenumbruch enthalten OHNE einen „escaped character“ wie „\n“ zu nutzen.

Lektion 57 – Übung 5 Beispiellösung

https://go.dev/play/p/P_HdKpJbE21

Lektion 58 – Übung 6

Erstellen Sie 4 Konstanten der nächsten Jahreszahlen beginnenend mit dem aktuellen Jahr und nutzen Sie dazu den Ausdruck iota in allen Wertzuweisungen. Geben Sie die Konstanten nebeneinander durch Leerzeichen getrennt aus.

Lektion 59 – Übung 6 Beispiellösung

https://go.dev/play/p/H0N99M4Zvml

Quiz 2

Lektion 60 – Übung 7 Quiz-2 Lösung

Lektion 61 – Kontrollstrukturen (control flow) - let it flow!

Kontrollstrukturen sind in der Informatik die Vorgabe, in welcher Reihenfolge die Handlungsschritte eines Algorithmus abgearbeitet werden. In imperativen Programmiersprachen werden sie durch **Kontrollanweisungen** (**Steuerkonstrukte**) implementiert. Mit Kontrollstrukturen können Programme auf verschiedene Zustände reagieren, indem Programmteile nur bedingt (bedingte Anweisung) oder wiederholt (Schleife) ausgeführt werden.[*]

Flow control (english): https://en.wikipedia.org/wiki/Control_flow

Kontrollstruktur(en) (deutsch): https://de.wikipedia.org/wiki/Kontrollstruktur

Lektion 62 – init, cond, post

Eine „For“-Schleife wird durch das Keyword „for“ eingeleitet. Sie ermöglicht es, einen mit {} umschlossenen Codeblock wiederholt auszuführen. „For“ sollte hier vielleicht nicht mit „für“ übersetzt werden, sondern gibt einen Zeitraum an. Die Bedeutung ist also eher „solange“. Also vereinfacht: Solange dies (noch) zutrifft, mache jenes.

Mit drei Literalen kann dieses „dies“, was zutreffen kann beschrieben werden.

1. **Init**ialisierung: eine Zählvariable wird für die Dauer der Schleifenausführung deklariert und mit einem Startwert initialisiert.

2. **Cond**ition/Bedingung: Es wird eine Abbruchbedingung für weitere Durchläufe gestellt.

Und

3. **Post**-increment oder -dekrement wird der Wert der Zählvariable verändert.

Einfaches Beispiel: https://go.dev/play/p/YjvKOOz9tbZ

Einfache Erklärungen: https://gobyexample.com/for

Beispiel, wie Pre-Inkrement/Decrement in Go umgesetzt werden muss: https://go.dev/play/p/STbMiE5IoAg

Für Fortgeschrittene nochmals hier: https://yourbasic.org/golang/gotcha-increment-decrement-statement/

Lektion 63 – Beispiel für verschachtelte Schleifen

Beispiel für eine einfach verschachtelte Schleife mit zwei For-Statements: https://go.dev/play/p/9aTSTdIxslk

Lektion 64 – Die For-Anweisung/Dokumentation verstehen

Beispiele und Definition von For-Anweisungen in dem Spezifikationen: https://go.dev/ref/spec#For

Erweiterte Backus-Naur Form zu Darstellung der Syntaxvorschriften in Programmiersprachen: https://de.wikipedia.org/wiki/Erweiterte_Backus-Naur-Form

Aus der Dokumentation der Programmierer/Hersteller von Go (ehemals „Effective Go“) mit einfachen Beispielen: https://go.dev/doc/effective_go#for

Lektion 65 – Break und Continue

Beispiele für Break um aus einer Endlosschleife herauszukommen:
https://go.dev/play/p/2j1M7rDnVAF

Beispiel, für die mögliche Anwendung von break und continue:
https://go.dev/play/p/GoNHeHQkRtI
und nochmals mit anders gestalteter Bedingung (Bit Operator):
https://go.dev/play/p/pvmEI74HKoB

Lektion 66 – ASCII Zeichen in Schleife ausgeben

Beispiel: https://go.dev/play/p/ttMWk2ccYDk

Lektion 67 – if - die bedingte Verzweigung

Beispiel: https://go.dev/play/p/arW3mnwYijt

Lektion 68 – if, else if, else - Wenn dies, dann jenes, ansonsten welches ...

Beispiel: https://go.dev/play/p/fE9OTBwRywG

Lektion 69 – for- und if-Statements mit Modulo Operator in einem Beispiel

Der Beispiel: https://go.dev/play/p/1qiFBsbcGWR

Lektion 70 – Switch Anweisung in Aktion

Das switch Statement / die switch Anweisung

```
switch / case / default
```

- „fall-through“ ist nicht standard, d.h. kein break nötig!
- fall-through aber möglich

- mehrere Fälle nacheinander
- Die Fälle können auch wieder testbare Ausdrücke sein (Fälle laufen, wenn „true“)

Beispiele:

switch mit bool Werten: https://go.dev/play/p/FPQp7-dfHgK

„fall-through“ ist nicht standard: https://go.dev/play/p/eQlPOXdwCF6

fall-through aber möglich: https://go.dev/play/p/CmJRvH6y5xu

default: https://go.dev/play/p/Ut_gfOCJzEe

switch für einen Wert, welcher läuft? https://go.dev/play/p/t__ffPEGEAB

switch für einen Wert, für den mehrere Fälle gleichzeitig geprüft werden: https://go.dev/play/p/J1vmeg5yPO-

Lektion 71 – switch - Blick in die Dokumentation

https://go.dev/ref/spec#Switch_statements

Lektion 72 – Logische Vergleichsoperatoren

```
fmt.Println(true && true)
fmt.Println(true && false)
fmt.Println(true || true)
fmt.Println(true || false)
fmt.Println(!true)
```

https://go.dev/play/p/LdJlmtdodwC

https://go.dev/play/p/1OmeFTU5lto

https://go.dev/ref/spec#Logical_operators

Lektion 73 – browsh - Beispiel für Go Progammierung

Website und Download: https://www.brow.sh und https://www.brow.sh/downloads/

Quellcode (in Klammern ein großer Teil des Go-Codes)

https://github.com/browsh-org/browsh
(https://github.com/browsh-org/browsh/tree/master/interfacer/src/browsh)

Lektion 74 – Übung 1

Geben Sie die Zahlen 1 bis 10000 aus.

Lektion 75 – Übung 1 Beispiellösung

https://go.dev/play/p/8AJQXsRhKgV

Alternative (unelegant): https://go.dev/play/p/BW_c0WEIvPc r

Lektion 76 – Übung 2

Geben Sie jeden „rune code point“ der Großbuchstaben des Aphabets drei mal aus. Das sollte so in etwa aussehen:

65

U+0041 'A'

U+0041 'A'

U+0041 'A'

66

U+0042 'B'r

U+0042 'B'

U+0042 'B'

… bis zum Character „Z“, also A bis Z

Lektion 77 – Übung 2 Beispiellösung

https://go.dev/play/p/gUFgee2tasg

Lektion 78 – Übung 3

Erstellen Sie eine Schleife mit

```
for condition { }
```

und geben Sie damit alle Jahre aus, die sie bereits leben (in denen sie gelebt haben).

Lektion 79 – Übung 3 Beispiellösung

https://go.dev/play/p/9SnkJMv-7FM

Lektion 80 – Übung 4

Erstellen Sie eine Schleife mit

```
for { }
```

und geben Sie damit alle Jahre aus, die sie bereits leben (in denen sie gelebt haben).

Lektion 81 – Übung 4 Beispiellösung

https://go.dev/play/p/Dg-aqljad-v

Lektion 82 – Übung 5

Geben Sie den Rest (modulo) aus, der bei der Teilung der Zahlen zwischen 10 und 100 (jeweils einschließlich) durch 4 ergibt.

Lektion 83 – Übung 5 Beispiellösung

https://go.dev/play/p/3_SNxzOcR4Q

Lektion 84 – Übung 6

Erstellen Sie ein Programm, das ein if-Statement nutzt.

Lektion 85 – Übung 6 Beispiellösung

https://go.dev/play/p/0fpYEJ_SksX

Lektion 86 – Übung 7

Erweitern Sie das Programm aus Übung 6, dass es nun auch “else if” und “else” nutzt.

Lektion 87 – Übung 7 Beispiellösung

https://go.dev/play/p/g-rzgO9gNKS

Lektion 88 – Übung 8

Erstellen Sie ein Programm, das ein switch statement ohne die explizite Angabe eines Ausdrucks nutzt.

Lektion 89 – Übung 8 Beispiellösung

https://go.dev/play/p/goHitje8O4b

Lektion 90 – Übung 9

Erstellen Sie ein Programm, das ein switch statement nutzt und einen Ausdruck vom Typ „String“ mit dem Namen „favSport“ abfragt. Geben Sie in den Fallunterscheidungen drei Sportarten und einen default case die Ausgabe „Sport interessiert mich nicht.“ an.

Lektion 91 – Übung 9 Beispiellösung

https://go.dev/play/p/idClavVetC6

Lektion 92 – Übung 10

Geben Sie diese logischen Vergleiche und ihre Ergebnisse aus:

```
fmt.Println(true && true)
fmt.Println(true && false)
fmt.Println(true || true)
fmt.Println(true || false)
fmt.Println(!true)
```

Lektion 92 – Übung 10 Beispiellösung

https://go.dev/play/p/8m41Yok0djz

Quiz 3

Lektion 94 – Übung 11 Quiz 3 gemeinsame Lösung

Lektion 95 – Array

Allgemeines über Arrays: https://de.wikipedia.org/wiki/Feld_(Datentyp)

und über Arrays in Go im Besonderen: https://go.dev/doc/effective_go#arrays

Beispiel Deklaration mit var: `var x [5]int`

Beispiel Deklaration und Wertzuweisung mit Short Declaration Operator (Syntax mit „dem zugrundeliegenden Typ“ vorangestellter Länge in eckigen Klammern und Wertelist in geschweiften Klammern für Kommata getrennt): `x := [5]int{1,2,3,4,5}`

Arrays in Go:

- eine Datenstruktur, um gleichartige Werte (desselben Typs) in eine Reihenfolge zu bringen und über einen Index ansprechbar zu machen.
- Arrays haben Werte, d.h. man kopiert alle Werte eines Arrays in ein anderes, nicht nur eine Referenz zum ersten Werte. (Call-by-Value)
- Die Länge eines Arrays (Anzahl der Elemente) ist Teil seines Typ, d.h. verschieden lange Arrays haben werden als von einem verschiedenen Typ betrachtet.

Beispiel: https://go.dev/play/p/u1HJlHmJrrJ

Lektion 96 – Composite literals

Beispiel eines composite literals anhang eines Slices: https://go.dev/play/p/RApQwMH_ZdM

Lektion 97 – Slices sind die besseren Arrays

Lektion 98 – Mit range über Slices iterieren

Beispiele range: https://go.dev/play/p/g1Y04Hkyily

Lektion 99 – Slice slicen - oder mal eine Scheibe abschneiden

Slice erstellen und ausgeben.

Slice mit Range ausgeben.

Slice **von** (Position eingeschlossen) **bis unter** (Position nicht mehr eingeschlosen) mit Doppelpunkt ausgeben (Operator „:“)

Slice von bis mit Ausdrücken testen.

Ursprüngliches Slice bleibt unangetastet, aber man kann den Inhalt einem neuen Slice zuweisen.

Beispiele Slicing a Slice: https://go.dev/play/p/6RUA6d0hjrJ

Lektion 100 – Append - etwas an ein Slice anfügen

Spezifikation und Beschreibung von `append()` mit interessanten Anwendungsbeispielen: https://go.dev/ref/spec#Appending_and_copying_slices

Beispiel für verschieden Nutzung von `append()`: https://go.dev/play/p/yTqL33RFZDL

Lektion 101 – Append-Paradox - Etwas aus einem Slice löschen

Es gibt in Go keine eingebaute „delete from slice“-Funktion, stattdessen, sollte man `append()` benutzen und das neue Slice aus dem Slice bis zu dem zu löschenden Element und den Elementen nach dem zu löchenden Element zusammensetzen:

Beispiel: https://go.dev/play/p/VI0z1wuNSsW

Lektion 102 – Slice erstellen mit make()

Overhead/Mehraufwand zur Ausführungszeit

Beispiel: https://go.dev/play/p/4FVqUinfKL1

Wenn menötigte Größe und Kapazität zur Compilezeit bekannt sind (oder zumindest gut abschätzbar), können und sollten slices mit der Funktion mak erstellt werden.

Make bei effective Go: https://go.dev/doc/effective_go#allocation_make

Syntax: `make(Typ, Länge, Kapazität)`

liefert zurück ein slices (!) mit den angeforderten eigenschäften, deren in Länge angegebenen Werte bereits mit Null-Werten (Zero Values des Typs) gefüllt sind.

Beispiel make: https://go.dev/play/p/SJidFQvo48f

Lektion 103 – Multidimensionale Slices

Slices können mehrere Dimenstionen haben.

Beispiel mit Slices aus mehreren Dimensionen aus eindimensionalen Slices (Strings): https://go.dev/play/p/B4Bn0weHQ0V

Lektion 104 – Map - eine Einführung, Komma Okay

Map ist ein eigener Datentyp in Go, der es ermöglicht, ungeordnete Listen mit Werten eines Typs (Element Type) anhand Werten eines (möglicherweise anderen) Schlüsseltyps (Key Type) zu durchsuchen.

Das Durchsuchen von Maps ist auch bei großen Datenmengen sehr effektiv und schnell.

Ist der gesuchte Schlüsselwert (key value) nicht in der map vorhanden, wird nil (Zero-Value des Element Typs zurückgeliefert. Aber Anfragen an Maps liefern ebenfalls einen Bool-Wert mit, der das Vorhandensein des Key Values bestätigt oder verneint.

Das ermöglicht das sogenannte „Komma Okay“-Konstrukt, das bei Abfragen von Keyvalues eine Unterscheidung zwischen „Null“ und „nicht vorhanden“ ermöglicht.

Beispiel map als Datentyp und dem in Englisch „comma okay idiom“ genannten Ausdruck: https://go.dev/play/p/qulOJQOx26O

Lektion 105 – Element in map einfügen und mit range darüber iterieren

Beispiel: https://go.dev/play/p/rZ-DcakRf1r

Lektion 106 – Element einer map entfernen mit delete()

Mit der Funktion `delete(map, KeyValue)` kann man einen Eintrag aus einer map entfernen.

Beispiel: https://go.dev/play/p/GcIgLiIEQU1

Lektion 107 – Übung 1

Benutzen Sie ein „composite literal“, um:

- ein Array mit 5 Elementen vom Typ int zu erzeugen
- Weisen Sie manuell jeder Indexposition einen Wert zu
- Nutzen Sie eine For-Schleife mit „range“ um das Array und Index auszugeben
- nutzen Sie eine formatierte Ausgabe

- und geben Sie anschließend den Typ des Arrays aus

Lektion 108 – Übung 1 Beispiellösung

Beispiel: https://go.dev/play/p/ZQENwYHdO5Q

Lektion 109 – Übung 2

Benutzen Sie ein „composite literal“, um:

- erstellen Sie ein Slice aus Werten vom Typ int
- weisen Sie 10 Werte zu
- nutzen Sie eine For-Schleife mit „range“ um Werte des Slices und Index auszugeben
- nutzen Sie formatierte Ausgabe
- und geben Sie anschließend den Typ des Slice aus

Lektion 110 – Übung 2 Beispiellösung

Beispiel: https://go.dev/play/p/SZJZ8mIKPnh

Lektion 111 – Übung 3

Erstellen Sie folgendes Slice mit Werten vom Typ int:

```
[42 43 44 45 46 47 48 49 50 51]
```

Nutzen Sie „Slicing“ um folgende Ausgaben zu erreichen (ohne das Slice zu verändern)

```
[42 43 44 45 46]
[47 48 49 50 51]
[44 45 46 47 48]
[43 44 45 46 47 78 49 50]
```

Lektion 112 – Übung 3 Beispiellösung

Beispiel: https://go.dev/play/p/a1bIYudIGni

Lektion 113 – Übung 4

Führen Sie folgende Schritte aus.

Beginnen Sie mit folgendem Slice:

```
x := []int{42, 43, 44, 45, 46, 47, 48, 49, 50, 51}
```

- Hängen Sie mit append() den Werte 51 an
- Geben Sie das slice aus

- Hängen Sie ein **einem** Statement die Werte 52, 53 und 54 an
- Geben Sie das slice aus
- Hängen Sie mit **einem** Statement an das Slice das folgende Slice an

```
y := []int{56, 57, 58, 59, 60}
```

Geben Sie das Slice x aus

Lektion 114 – Übung 4 Beispiellösung

Beispiel: https://go.dev/play/p/tlJjoczViim

Lektion 115 – Übung 5

Führen Sie nachfolgende Schritte aus.

Beginnen Sie mit folgendem Slice:

```
x := []int{42, 43, 44, 45, 46, 47, 48, 49, 50, 51}
```

Nutzen Sie `append()` und Slicing, um das folgende Slice dem neu zu erstellenden Slice y zuzuweisen:

```
[42, 43, 44, 48, 49, 50, 51]
```

Lektion 116 – Übung 5 Beispiellösung

Beispiel: https://go.dev/play/p/Pg1d724mtUG

Lektion 117 – Übung 6

Erstellen Sie ein Slice, um die Namen aller deutschen Bundeländer zu speichern. Verwenden Sie `make()` und `append()`, um dies zu tun.

Ziel: Das Array, das dem Slice zugrunde liegt, soll nicht mehr als einmal erstellt wird.

Wie lang ist Ihr Slice? Wie groß ist die Kapazität?

Geben Sie alle Werte zusammen mit mit ihrer Indexposition aus, ohne „range“ zu verwenden.

(Aufgabe ist weniger einfach als sie aussieht.)

Hier die Bundesländer:

```
`Bayern`, `Baden-Württemberg`, `Berlin`, `Brandenburg`, `Bremen`,
`Hamburg`, `Hessen`, `Mecklenburg-Vorpommern`, `Niedersachsen`,
`Nordrhein-Westfalen`, `Rheinland-Pfalz`, `Saarland`, `Sachsen`,
`Sachsen-Anhalt`, `Schleswig-Holstein`, `Thüringen`
```

Lektion 118 – Übung 6 Beispiellösung

Beispiel: https://go.dev/play/p/SfafU8z6ypC (aus Video) oder https://go.dev/play/p/TMKXkElrsEL (auch gut)

Das geht nicht: https://go.dev/play/p/0DkjJnMcQnv

Lektion 119 – Übung 7

Erstellen Sie ein Slice von Slice von String ([][]string). Speichern Sie die folgenden Werte:

```
"James", "Bond", "Bond, James Bond"
```

```
"Papa", "Schlumpf", "Schlumpf, Papa Schlumpf"
```

```
"Rick", "Sanchez", "Schlauster Kopf im Multiversum"
```

```
"Morty", "Smith", "Hauptberuflicher Sidekick"
```

Nutzen Sie Sie zwei ineinander verschachtelte For-Schleifen mit range die Nummer (Index) des Slices auszugeben und jeweils darunter alle Werte des jeweiligen Slice mit vorangestellter Position innerhalb des jeweiligen Slice.

In Etwa:

Slice Nummer: 0

Postion 0: James

Postion 1: Bond

Postion 2: Bond, James Bond

usw ...

Lektion 120 – Übung 7 Beispiellösung

Beispiel: https://go.dev/play/p/uF-QO1H-oD0

Lektion 121 – Übung 8

Erstellen Sie eine map mit einem Key vom Typ string, der "Vorname Familienname" einer Person entspricht, und einem Werten vom Typ []string, die ihre Lieblingsdinge speichert. Speichern Sie sieben Datensätze in Ihrer map. Geben Sie alle Werte aus, zusammen mit ihrer Indexposition im Slice aus.

```
`Stan Smith`, `Amerika`, `Familie`, `Jesus`
```

```
`Francine Smith`, `Lippenstift`, `Pinke Kleider`, `Weinen unter der Dusche`
```

```
`Hayley Smith`, `Stirnband`, `Tank Top`, `Sandalen`
```

```
`Steve Smith`, `Computer`, `Mädchen`, `Freunde`
```

```
`Roger Smith`, `Fernsehen`, `Alkohol`, `Drogen`
```

```
`Klaus Heissler`, `Skispringen`, `Schwimmen`, `Rap & Hip Hop`
`Jeff Fischer`, `Gras rauchen`, `Fish (die Band)`, `seinen Hut`
```

Lektion 122 – Übung 8 Beispiellösung

Beispiel: https://go.dev/play/p/-vIAJzWJ2Ko

Lektion 123 – Übung 9

Aufbauend auf dem Code aus der vorherigen Übung fügen Sie einen Eintrag für sich selbst der Liste hinzu. Geben Sie die gesamten Werte mit einer For-Schleife und mit der Nutzung von range aus.

Lektion 124 – Übung 9 Beispiellösung

Beispiel: https://go.dev/play/p/0SgLkOxKkeW

Lektion 125 – Übung 10

Entfernen Sie den Eintrag von Klaus aus der Map und stellen Sie dabei sicher, dass der Eintrag nur gelöscht wird, wenn er auch existiert!

Geben Sie die gesamten Werte mit einer For-Schleife und mit der Nutzung von range aus.

Lektion 126 – Übung 10 Beispiellösung

Beispiel: https://go.dev/play/p/0dWg8GA6QGP

Quiz 4

Lektion 127 – Übung 11 Quiz 4 gemeinsame Lösung

Lektion 128 – Structs - bringen Struktur ins Leben

Structs bieten die Möglichkeit, zusammengesetzte Strukturen aus aus verschiedenen Datentypen zu bilden und Variablen als Wert zuzuweisen. Das hat schon viel von Objekten und Klassen aus anderen Programmiersprachen, trotzdem sprechen wir in Go von „Values of Type“,also von Werten eines bestimmten Typs.

In der Regel werden diese structs mit dem keyword type einer eigenen Datentyp zugewiesen und zu Beginn inititalisiert, wenn eine Wertzuweisung erfolgen soll.

Erfolgt eine Wertzuweisung nach der definition eines Strucs (nachflegend „structure“) in der Form

```
x := structure{}
```

oder mit

`var x structure`

wird für alle Elemente des Structs eine Nullwert (Zero Value) angenommen. Das gilt auch für in einer Initialisierung ausgelassene Wertzuweisungen.

Beispiel: https://go.dev/play/p/Pc3Ocg1CwsO

Lektion 129 – Eingebettete structs

Struct können andere Structs als Element beinhalten. Bei der Deklaration ist eine Typangabe des inneren Structs nicht nötig, bei der Initialisierung schon. Auf Elemente von Structs kann man wie gewohnt mit dem Punktoperator („.“) zugreifen. Dabei ist es nicht zwingend nötig, Elemente eingebetteter Structs mit in der Hirarchie des Ausdrucks zu nennen. Ein Aufruf wie `aeusseresStruct.elementInneresStruct` reicht in der Regel.

Um Namenskollisionen zu vermeiden ist es aber möglich, sie mit `aeusseresStruct.InneresStruct.elementInneresStruct` anzusprechen.

Beispiel: https://go.dev/play/p/_MXt1YaUtXN

Lektion 130 – Blick in die Dokumentation

Structs: https://go.dev/ref/spec#Struct_types

Schöne Erklärung in Deutsch: https://geekflare.com/de/structs-in-golang/

Lektion 131 – Anonymous Structs - Structs ohne Namen

Anonyme Structs/anonymous structs: https://go.dev/play/p/2NbNPlrBOJv

Lektion 132 – Nachgang: Aufräumen (Zusammenfassung)

Ease of Programming (var x int, type indentifier struct{})

Genau bleiben und möglichst gut lesbaren Code schreiben. Go erlaubt „Abkürzungen“, aber zu gutem Coding gehört auch verständlicher und menschenlesbarer Cod

Ist Go eine objekt-orientierte Programmiersprache?

https://en.wikipedia.org/wiki/Object-oriented_programming

https://go.dev/doc/faq#Is_Go_an_object-oriented_language

Die strenge Typisierung von Go erfordert es, sich über Typen seiner Variablen immer im Klaren zu sein. https://go.dev/play/p/lm1j6KBc9Gp

Stöbern Sie in der Dokumentation und in den zahlreichen Angeboten von und über Go: https://go.dev/ und https://go.dev/doc/

Sie haben bis hierher bereits genug Kenntnisse gesammelt, um selbstständig Lern-Angebote zu suchen und wahrzunehmen, wie einfache Tutorials.

Lektion 133 – Übung 1

Erstellen Sie Ihren eigenen Datentyp "person", der einen zugrundeliegenden Typ "struct" hat, so dass die folgenden Daten speichern kann:

- Vorname
- Nachname
- Alter
- mehrere Lieblings-Eiscreme-Sorten

Erstellen Sie zwei Werte vom Typ person. Geben Sie die Werte mittels range aus, die sich in einem Element vom Typ []string für die Lieblings-Eiscreme-Sorte angegeben sind.

Lektion 134 – Übung 1 Beispiellösung

Beispiel: https://go.dev/play/p/2YSo6VD1Q_m

Lektion 135 – Übung 2

Nehmen Sie den Code aus der vorherigen Übung und speichern Sie die Werte vom Typ person in einer Map mit dem Schlüssel des Typs der den Nachnamen enthält. Greifen Sie auf jeden Wert in der Map zu und geben Sie auch die Werte aus, die im Slice enthalten sind.

Lektion 136 – Übung 2 Beispiellösung

Beispiel: https://go.dev/play/p/L2kYQag99pL

Lektion 137 – Übung 3

Erstellen Sie einen neuen Typ: fahrzeug.

- Der zugrunde liegende Typ ist ein struct.
- Die Felder:
  - anzahlTüren
  - farbe
- Erstellen Sie zwei neue Typen: lkw und pkw
- Der zugrunde liegende Typ jedes dieser neuen Typen ist ein struct.
- Betten Sie den Typ "fahrzeug" in lkw und pkw ein.
- Geben Sie dem lkw das Feld "vierrad", das auf bool gesetzt wird.
- Geben Sie der pkw das Feld "luxus", das auf bool gesetzt wird.

Verwendung der fahrzeug-, lkw- und pkw-Structs:

- Erstellen Sie mit einem Composite Literal „brummi“ und weisen Sie den Felder Werte zu.
- Erstellen Sie mithilfe eines Composite Literal „mittelklassewagen“ und weisen Sie den Felder Werte zu.
- Geben Sie diese beiden „Values of Type“ aus.
- Geben Sie wenigstens einen der Werte der eingebetteten Felder aus.

Lektion 138 – Übung 3 Beispiellösung

Beispiel: https://go.dev/play/p/ekzLAkKHYLG

Lektion 139 – Übung 4

Erstellen Sie ein „anonymous“ struct, also ein Struct ohne Identifier.

Lektion 140 – Übung 4 Beispiellösung

Beispiel: https://go.dev/play/p/NP_ClQR_6X_4

Quiz 5

Lektion 141 – Übung 5 Quiz 5 gemeinsame Lösung

Lektion 142 – Funktionen - Syntax

Syntax allgemein:

```
func (r receiver)identifier(parameter)(return(s)){ code }
```

Beispiel einfache Funktion ohne Rückgabe von Werten: https://go.dev/play/p/5H5PLpHHRq-

Beispiel einer Funktion mit Übergabe von einem Argument: https://go.dev/play/p/O5x2PRGQxhd

Beispiel einer Funktion mit Übergabe von einem Argument und Rückgabe eines Wertes: https://go.dev/play/p/JOj3C0V-8nf

Beispiel einer Funktion mit Übergabe von zwei Argumenten und Rückgabe zweier Werte: https://go.dev/play/p/fRMG45v8Nck

Lektion 143 – Variatische Parameter, die Zweite

Beispiel für die Übergabe von Argumenten gleichen Typs als variatische Parameter (variadic parameters) in eine Funktion: https://go.dev/play/p/6sgwi8G8abG und https://go.dev/play/p/UMnPRJkkdC1 und https://go.dev/play/p/jOQT1IVOXtk

Beispiel Übergabe Parameter verschiedenen Typs zusammen mit anschließendem variatischen Parameter: https://go.dev/play/p/15aBLS-rBKV

Main Takeaway:

- Variatische Parameter können in einer Funktionssignatur dazu diesen, eine Liste von beliebig vielen Werten gleichen Typs an eine Funktion zu übergeben, die innerhalb der Funktion als Slice dieser Werte zur Verfügung steht.
- Variatische Parameter werden mit dem ...-Operator angegeben und können nur als einmal und als letzter Parameter in der Signatur einer Funktion angegeben werden.

Beispiel für Anwendung von variatischen Parameter zur Übergabe von Argumenten and Funktion zur Ausummierung von Summanden beliebiger Anzahl innerhalb einer Funktion: https://go.dev/play/p/rnh1uJ3RtOI

Lektion 144 – Ein Slices "abrollen"

Specs: https://go.dev/ref/spec#Passing_arguments_to_..._parameters

Beispiel ein Slice „abrollen“ und die Werte als Reihe von Werten an eine Funktion übergeben: https://go.dev/play/p/Fq--KsU9PUe

Lektion 145 – Defer - Verzögerungstaktik

Das Keyword `defer` dient dazu die Ausführung von Code (in Funktionen) bis zu dem Zeitpunkt zu verzögern, in dem der umschließende Codeblock beendet wird oder „abstürzt“, bzw im Begriff ist abzustürzen („panicking“).

Beispiel: https://go.dev/play/p/pIgNKl2lNB9

Lektion 146 – Methods - Funktionen haben Methode(n)

Einfaches Beispiel einer Methode: https://go.dev/play/p/PL8aFcqSJRD

Können Funktionen/Methoden in Go mehrere Receiver haben? https://www.iops.tech/blog/method-receiver-types-in-go/ (und anderes Interessantes zu Methoden)

Einfaches Beispiel zu Methoden und Beispiel für die Nutzung von Pointern zu Structs in Methoden (als Receiver in Funktionen): https://gobyexample.com/methods

Das Beispiel kopiert aus dem vorherigen Link: https://go.dev/play/p/6cHAxhv9-uq

Lektion 147 – Methods – die Zweite

Ein etwas realitätsnaheres Beispiel für eine Methode in Go: https://go.dev/play/p/4qFmPRzGhAy

Lektion 148 – Methods – die Dritte - Call by Value / Call by Reference

Go spricht nicht gerne von “Call by Value“ und „Call by Reference“, sondern spricht ausschließlich von „Call by Value“, denn auch ein Pointer ist letztenendes ein Wert, nur eben vom Typ Pointer. Die Betrachtungsweise ist vereinfacht und vermeidet Ungenauigkeiten oder Missverständnisse.

Beispiel Methode „Call by value“: https://go.dev/play/p/kBase8A58Uk

Beispiel Methode „Call by Reference“: https://go.dev/play/p/XJ9KGFCHjyO (Pointer)

Lektion 149 – Einschub – Bleiben Sie dran

Lektion 150 – Interfaces und Polymorphismus I

Ausgangsbeispiel („Cleaning“ notwendig) : https://go.dev/play/p/kBase8A58Uk

Beispiel für Nutzung Interfaces: https://go.dev/play/p/AUrXgm2ehcG

Beispiel switch je nach type: https://go.dev/play/p/fdilwQLKbbL

Lektion 151 – Interfaces und Polymorphismus II

Beispiel erweitert mit „Assertion“: https://go.dev/play/p/nmlrE0lsgWk

Bill Kennedy Blog über „Composition und Nutzung von Interfaces in Go“:

https://www.ardanlabs.com/blog/2015/09/composition-with-go.html

Lektion 152 – Interfaces reloaded

Unser Beispiel: https://go.dev/play/p/-m3l05i15wl

Inspiriert von GoByExample-Beispiel: https://gobyexample.com/interfaces

Lektion 153 – Interfaces Revolutions

Unser Beispiel: https://go.dev/play/p/pAsF3ARAsdk

Inspiriert von Jordan Orelli-Beispiel: https://jordanorelli.com/post/32665860244/how-to-use-interfaces-in-go

Lektion 154 – Anonyme Funktionen - Sie brauchen keinen Namen

Beispiel für anonyme Funktionen: https://go.dev/play/p/t2PBbF3yaW3

Lektion 155 – func Ausdrücke - auf geht's in den Kaninchenbau

Beispiel für Funktionen als Ausdrücke: https://go.dev/play/p/eY3mQzUIl4a

Lektion 156 – Eine Funktion als Rückgabewert

Wiederholung Funktionen/Anonyme Funktionen: https://go.dev/play/p/TtVOZTk2PK0

Wiederholung Funktionen als Ausdrücke/Anonyme Funktionen:
https://go.dev/play/p/0aQHGRpLnFO

Neu: Funktionen können auch als Typ für Rückgabewerte dienen:
https://go.dev/play/p/4wMJwdIKmMe

Lektion 157 – Callbacks – Funktionen als Parameter anderer Funktionen

Eine Callback-Funktion bezeichnet in der Informatik eine Funktion, die einer anderen Funktion als Parameter übergeben und von dieser unter definierten Bedingungen und Argumenten aufgerufen wird.

Beispiel einfacher Callbacks: https://go.dev/play/p/O9zBEEyHIoA

Lektion 158 – Closure

Eine Closure-Funktion bezeichnet man eine anynome Funktion, die durch eine andere Funktion zurückgeliefert wird und während ihrer Erstellung Zugriff auf einen Wert während der Erstellung bekommt (Context). Außerhalb der der Closure-Funktion ist dieser Wert nicht erreichbar.

Einfaches Beispiel für Closure-Funktion: https://go.dev/play/p/VTeMIjLqInY

Lektion 159 – Rekursion

Als Rekursion bezeichnet man eine Funktion, die während des Ablaufes eine Kopie seiner selbst aufruft.

Beispiel Fakultät: https://go.dev/play/p/sdj9_78nN0V

Lektion 160 – Kurze Wiederholung (und Tipp gegen Prokrastination)

Themen, die man jetzt konzeptionell verstanden haben sollte:

- Funktionen
- Zweck von Funktionen
  Codeabstraktion, Wiederverwendung
- func, receiver, identifier, params, returns

- parameters vs arguments
- Variatische Funktionen

  Mehrfache “variatische” Parameter
  Mehrfache “variatische” Argumente
- Rückgabewerte (returns)
  Mehrfache returns
  „Benannte“ returns (irgednwie komisch)
- Funktionaausdrücke
  einer Variablen eine Funktion zuweisen und sie damit vom Typ Funktion machen!
- Callbacks
  eine Funktion einer Anderen funktion als Argument übergeben
- closure
  den Geltungsbereich einer Variablen in einer anderen Variablen speichern und wird so nur im inneren Geltungsbereich sichtbar.
- Recursion
  Funktionen, die sich selbst aufrufen Beispiel: Fakulatät
- interfaces und „leeres Interface“ (`interface{}`)

Wichtiger Hinweis: **Focus on what’s important; not upon what’s urgent.**

Lektion 161 – Übung 1

In dieser Übung:

- Erstellen Sie eine Funktion mit dem Identifier „foo“, die einen Wert vom Typ int zurückgibt
- Erstellen Sie eine Funktion mit dem Identifier „bar“, die einen Wert vom Typ int und einen einen Wert vom Typ string zurückgibt
- Erstellen Sie für beide funktionen einen gültigen Funktionsaufruf und weisen sie neu erstellten Variablen die Rückgabewerte zu.
- Geben Sie die Werte aus.

Lektion 162 – Übung 1 - Beispiellösung

Beispiellösung: https://go.dev/play/p/XXIlbJorkdt

Lektion 163 – Übung 2

Erstellen Sie eine Funktion mit dem Identifier „foo“, die

- die einen variatischen Parameter vom Typ int entgegennimmt
- Übergeben Sie einen Wert vom Typ []int in adequater Weise an die Funktion

- Geben Sie eine Summe aller übergebenen Eingabewerte als Rückgabewert zurück

Erstellen Sie eine Funktion mit dem Identifier „bar“, die

- als Parameter Werte vom Typ []int entgegennimmt
- Geben Sie eine Summe aller übergebenen Eingabewerte als Rückgabewert (Typ int) zurück

Lektion 164 – Übung 2 - Beispiellösung

Beispiellösung: https://go.dev/play/p/ZXZJI6g6SYD

Lektion 165 – Übung 3

Erstellen Sie zwei Funktionen und rufen Sie sie auf. Sorgen Sie mit „defer“ dafür, dass der erste Aufruf bis nach den zweiten Aufruf verzögert wird.

Lektion 166 – Übung 3 - Beispiellösung

Beispiellösung: https://go.dev/play/p/C6l_iJZJc4W

Lektion 167 – Übung 4

Erstellen Sie einen Typ mit unterliegendem Struct mit dem Identifier „person“

Wählen Sie angemessene Typen für die Elemente:

    vorname

    nachname

    alter

- weisen Sie dem Typ „person“ eine Methode zu, die den Identifier „sagt“ hat.

  Die Methode greift auf das in „person“ definierte Struct zu und gibt einen String mit Name und Alter aus.
- Erstellen Sie einen Wert des Typs „person“.
- Rufen Sie die Methode für den Werte auf.

Lektion 168 – Übung 4 - Beispiellösung

Beispiellösung: https://go.dev/play/p/exrn0K6YayF

Lektion 169 – Übung 5

Erstellen Sie einen Typ „quadrat“ und einen Typ „kreis“ beruhend auf structs.

Erstellen Sie eine Method „fläche“, die einen Wert vom Typ float64 zurückgibt und weisen Sie beiden Typen die Methode zu.

Fläche eines Kreisen = π r 2

Fläche eines Quadrates = Seitenlänge * Seitenlänge

Erstellen Sie einen Typ „formen“, der ein Interface definiert, das durch die Implementierung der Methode „fläche“ definiert ist.

Erstellen Sie eine Funktion mit dem Identifier „info“, die den Typ „formen“ entgegennimmt und die Fläche ausgibt.

Erstellen Sie, bzw geben Sie aus mittels „info“:

Wert vom Typ „quadrat“.

Wert vom Typ „kreis“.

Rufen Sie die Funktion info für beide Werte auf!

Lektion 170 – Übung 5 - Beispiellösung

Beispiellösung: https://go.dev/play/p/mJ68SZjtuA_7

Lektion 171 – Übung 6

Erstellen und nutzen Sie eine „anonyme“ Funktion.

Lektion 172 – Übung 6 - Beispiellösung

Beispiellösung: https://go.dev/play/p/nUzR8cM4ZS3

Lektion 173 – Übung 7

- Weisen Sie eine Variablen eine Funktion zu, (die irgendetwas tut) und rufen Sie diese Funktion auf.
- Erweitern Sie das Beispiel so, dass die Funktion auch einen Wert als parameter entgegennimmt und ausgibt.

Lektion 174 – Übung 7 - Beispiellösung

Beispiellösung: https://go.dev/play/p/PEUmGJDBhVr und https://go.dev/play/p/bDyC0rbSDk9

Lektion 175 – Übung 8

- Erstellen Sie eine Funktion, die eine Funktion als Rückgabewert liefert
- Weisen Sie die zurückgegebene Funktion einer Variablen zu
- Rufen Sie die Funktion durch die Variable auf.

Lektion 176 – Übung 8 - Beispiellösung

Beispiellösung: https://go.dev/play/p/0_MLYRTWkOL

Lektion 177 – Übung 9

Erstellen Sie einen Callback, d.h. erstellen Sie eine Funktion, die eine Funktion (und ein einen Funktionswert) als Parameter entgegennimmt. Dann übergeben Sie eine Funktion (die z.B. einen String ausgibt) und einen Wert (zum Beispiel „You shall not pass!“) und bringen Sie zur Ausführung.

Lektion 178 – Übung 9 - Beispiellösung

Beispiellösung: https://go.dev/play/p/Wvti0LdwWZs

Lektion 179 – Übung 10

Closures „verkapseln“ den Geltungsbereich einer Variablen in einem Codeblock. Erstellen Sie eine Funktion mit dem Identifier „undNochEinEis“, die innerhalb einer zurückgelieferten anonymen Funktion eine Variable „soVieleEis“ hochzählt.

Erstellen Sie eine Variable „spongbob“ und eine Variable „patrick“, der Sie jeweils die Funktion „undNochEinEis“ zurodnen. Rufen Sie die „spongbob“-Funktion drei mal auf, die „patrick“-Funktion dreiundzwanzig mal.

Lektion 180 – Übung 10 - Beispiellösung

Beispiellösung: https://go.dev/play/p/F3tvqkxgXaP

Quiz 6

Lektion 181 – Übung 11 Quiz 6 gemeinsame Lösung

Lektion 182 – Konzept Speicher simplifiziert…

Lektion 183 – Pointer - das unbekannte Wesen!

Beispiel für Pointer, Deklaration, (De)Referenzierung, Typzuweisung: https://go.dev/play/p/W37yF4jdCCq

Lektion 184 – Wann und wie man Pointer einsetzt

Simples Beispiel ohne Pointer: https://go.dev/play/p/cYJHFNdqJMi

Simples Beispiel mit Pointer: https://go.dev/play/p/x1KlPBLBz9t

Beispiel mit Pointer und zusammengesetzem Datentyp (struct): https://go.dev/play/p/lpwY9tak_XG

Mutation / to mutate = einen Wert ändern (mittels eines Pointers, der auf die Stelle im Speicher zeigt, an dem der Wert liegt).

Lektion 185 – Method Sets / Sätze von Methoden

Der Methodensatz eines Typs bestimmt die Methoden, die auf einen Operanden dieses Typs angewendet werden können. Jeder Typ hat einen (möglicherweise leeren) Methodensatz, der ihm zugeordnet ist:

- Der Methodensatz eines definierten Typs T besteht aus allen Methoden, die mit dem Receivertyp T deklariert sind.
- Der Methodensatz eines Zeigers auf einen definierten Typ T (wobei T weder ein Zeiger noch ein Interface ist) ist die Menge aller Methoden, die mit Empfänger *T oder T deklariert sind.
- Die Methodenmenge eines Interfacetyps ist die Schnittmenge der Methodenmengen jedes Typs in der Typenmenge des Interfaces (die resultierende Methodenmenge ist in der Regel nur die Menge der deklarierten Methoden im Interface).

Weitere Regeln gelten für Structs (und Zeiger auf Structs), die eingebettete Felder enthalten, wie im Abschnitt über Struct-Typen beschrieben. Jeder andere Typ hat einen leeren Methodensatz.

In einem Methodensatz muss jede Methode einen eindeutigen, keine Leerzeichen enthaltenden Methodennamen haben.

In eigenen Worten:

Methodensätze legen fest, welche Methoden einem Typ zugeordnet sind. Damit sind sie genau, was ihre Bezeichnung aussagt: Die Menge alle Methoden, die ein Typ implementiert: Was ist die Menge der Methoden eines bestimmten Typs? Das ist sein Method Set.

2 Fälle:

1. Receiver ist kein Pointer

    kann Werte als Pointer oder nicht Pointer entgegennehmen!

2. Receiver ist ein Pointer (eines Typs)

    kann ausschließlich Werte in Form von Pointern entgegennehmen

```
Receivers      Werte
---------------------------
(t T)          T oder *T
(t *T)         *T
```

Daraus ergeben Sich vier (+1) Fälle, die wir durchtesten sollten:

Receiver und Wert sind keine Pointer: https://go.dev/play/p/xsj8eEIelqT

Receiver kein Pointer, aber Wert als Pointer (eine Adresse) übergeben: https://go.dev/play/p/UxVgzwXgWW3

Receiver und Wert sind beide Pointer: https://go.dev/play/p/UpIsJaflj_f

Receiver ist Pointer, aber Wert ist keiner: https://go.dev/play/p/QWTlg-0vIJF **(Compilieren schlägt fehl!)**

Aber dieser Code funktioniert, beachten Sie den Unterschied durch den Einsatz von Interface!

https://go.dev/play/p/XgAbSmSJHbH

Lektion 186 – Übung 1

Erstellen Sie einen Wert und weisen sie ihn einer Variablen zu.

Geben Sie die Adresse aus, an der der Wert gespeichert ist.

Lektion 187 – Übung 1 - Beispiellösung

Beispiellösung: https://go.dev/play/p/3FipO1lv5T1

Lektion 188 – Übung 2

Erstellen Sie einen Typ (Identifier „person“) in Form eines Structs mit den Elementen

vorname	string
nachname	string
alter	int
adresse	string

Erstellen Sie eine Funktion „ändern“ mit *person (Pointer zu Typ person) als Parametertyp.

In der Funktion ändern Sie den Wert, der unter *person im Element adresse gespeichert ist.

Wichtig: Um das struct.element zu dereferenzieren nutzen Sie: (*value).field

`p1.adresse` und `(*p1).adresse` sollten equivalent sein, weil:

> ***“As an exception, if the type of x is a named pointer type and (*x).f is a valid selector expression denoting a field (but not a method), x.f is shorthand for (*x).f.”***
> https://go.dev/ref/spec#Selectors

In Funktion `main()`“

Erstellen Sie einen gültigen Wert vom Typ person.

Geben Sie den Wert aus.

Rufen Sie “ändern” mit Übergabe des korrekten Parameters auf, um den Wert zu ändern.

Geben Sie den Wert aus.

Lektion 189 – Übung 2 - Beispiellösung

Beispiellösung: https://go.dev/play/p/I2RRnQSq3sC

Quiz 7

Lektion 190 – Übung 3 Quiz 7 gemeinsame Lösung

Lektion 191 – JSON Package Dokumentation

Information über Packages, die in Go zur „Standard Library“ gehören finden wir unter:

https://pkg.go.dev/std

Hier finden wir in verschiedenen Kategorien (wo nötig) für Packages, die mit Go „mitgeliefert“ werden, d.h. die in Go selbst implementiert wurden und zum Import zur Verfügung stehen.

Zum Beispiel das Paket json aus der Kategorie „encoding“ finden wir unter
https://pkg.go.dev/encoding/json

Index: https://pkg.go.dev/encoding/json#pkg-index

Beispiel: https://pkg.go.dev/encoding/json#pkg-examples

Funktionen: https://pkg.go.dev/encoding/json#pkg-functions

Typen: https://pkg.go.dev/encoding/json#pkg-types und sogar der offenliegende Quellcode: https://pkg.go.dev/encoding/json#section-sourcefiles bis runter zur letzten Funktion in veschiedenen Teilen von Go (hier beispielhaft: https://cs.opensource.google/go/go/+/go1.18.1:src/encoding/json/encode.go bis ins kleinste Detail dokumentiert und mit tiefreichenden Kommentaren versehen.)

Weitere allgemeine und Go-spezifische Informationsquellen zu JSON

https://de.wikipedia.org/wiki/JavaScript_Object_Notation

https://eager.io/blog/go-and-json/

https://yourbasic.org/golang/json-example/
https://medium.com/go-walkthrough/go-walkthrough-encoding-json-package-9681d1d37a8f
https://golang.org/pkg/encoding/json/#Marshal

Lektion 192 – JSON marshal

https://pkg.go.dev/encoding/json#Marshal

Beispiel: https://go.dev/play/p/gdnH5NfvQLo

Lektion 193 – JSON unmarshal

https://pkg.go.dev/encoding/json#Unmarshal

Beispiel: https://go.dev/play/p/cs3IWoieS9O

Lektion 194 – Writer Interface

https://pkg.go.dev/std

https://go.dev/play/p/J9U-LZ0BE3O

Lektion 195 – Sortieren

Code um damit zu Starten: https://go.dev/play/p/LZYf3y_frPt

Code zum Sortieren: https://go.dev/play/p/-xs5jS4vKmd

Lektion 196 – Sortieren - diesmal an die eigenen Bedürfnisse angepasst

Code, um damit zu starten: https://go.dev/play/p/AVSEoTNDY83

Beispiel, sortiert nach Alter und nach Name: https://go.dev/play/p/oVfG9uUwLR7

Lektion 197 – bcrypt

Weiteres zu bcrypt: https://de.wikipedia.org/wiki/Bcrypt#Sicherheit

Notwendig, falls Sie das Beispiel in einer eigenen Entwicklungsumgebung laufen lassen wollen:

```
go get golang.org/x/crypto/bcrypt
```

```
go get -u golang.org/x/crypto/bcrypt
```

```
go env -w GO111MODULE=off
```

```
(go env -w GO111MODULE=auto) (für’s Zurückstellen)
```

Komplettes Beispiel mit Passwortumwandlung in einen Hashwert mittel bcrypt und anschließenden Vergleich eines Passwortes mit dem Hashwert (vom Go Playground):
https://go.dev/play/p/DUOgig77SqE

Lektion 198 – Übung 1

Nehmen Sie den folgenden Code als Grundlage: https://go.dev/play/p/wgCYvv88GnY

Packen Sie den `[]user` in JSON und geben Sie dieses aus.

Hilfe: Denken Sie daran, was Sie tun müssen, um eine Variable außerhalb ihres Pakets verfügbar zu machen!

Lektion 199 – Übung 1 - Beispiellösung

Beispiellösung: https://go.dev/play/p/ftKncctuYJi

Lektion 200 – Übung 2

Sie erhalten nach dem Funktionsaufruf zum Abschalten eines Antiviren-Programmes folgende Rückmeldung in JSON:

```
{"action":"stop","beta":false,"error":
{"code":0},"finished":true,"language":"enu","last_stage":"stopped"
,"package":"AntiVirus","pid":28386,"scripts":
[{"code":0,"message":"","type":"stop"}],"stage":"stopped","status"
:"stop","status_description":"translate from systemd
status","success":true,"username":"","version":"1.5.3-3077"}
```


Übertragen die die enthaltenden Daten in ein geeignetes Struct in Go und geben Sie anschließend den Boolean-Wert aus, der den Erfolg anzeigt.

Hilfe: https://mholt.github.io/json-to-go/

Lektion 201 – Übung 2 - Beispiellösung

Beispiellösung: https://go.dev/play/p/xL8mCejQmB7

Lektion 202 – Übung 3

Nehmen Sie den folgenden Code als Grundlage: https://go.dev/play/p/EMjjGmbZY4_Z

„Encoden“ sie den Wert vom Typ `[]user` in JSON und senden Sie das Ergebnis an `Stdout`.

Hilfe: Nutzen Sie json.NewEncoder(os.Stdout).encode(v interface{})

Lektion 203 – Übung 3 - Beispiellösung

Beispiellösung: https://go.dev/play/p/thS8nWmIjvH

Lektion 204 – Übung 4

Nehmen Sie den folgenden Code als Grundlage: https://go.dev/play/p/3tBoyBfzyLv

Sortieren die `[]int` und `[]string` für alle Benutzer.

Lektion 205 – Übung 4 - Beispiellösung

Beispiellösung: https://go.dev/play/p/HxBbH3yFtWc

Lektion 206 – Übung 5

Nehmen Sie den folgenden Code als Grundlage: https://go.dev/play/p/yzI9s_gdhyP

Sortieren Sie []user nach

- Name
- Alter

Sortieren Sie jeden []string „Sprüche“ jeden Users aphabetisch.

Geben Sie alles übersichtlich aus.

z.B. ähnlich:

Name, Alter

Spruch1

Spruch2

Spruch3

usw...

Lektion 207 – Übung 5 - Beispiellösung

Beispiellösung: https://go.dev/play/p/K8VANhsNI86

Lektion 208 – Nebenläufigkeit versus Parallelverarbeitung

Go ist die erste Programmiersprache, die nach der breiten Einführung von Mehrprozessorsystemen als Programmiersprache mit besonderen Fähigkeiten zur Parallelverarbeitung, entwickelt wurde.

https://de.wikipedia.org/wiki/Parallelrechner

https://de.wikipedia.org/wiki/Mehrprozessorsystem

https://de.wikipedia.org/wiki/Mehrkernprozessor#Einf%C3%BChrung

https://de.wikipedia.org/wiki/Go_(Programmiersprache)#Geschichte

https://de.wikipedia.org/wiki/Go_(Programmiersprache)#Nebenl%C3%A4ufigkeit

30 minuten, Rob Pike über Nebenläufigkeit und Parallelverarbeitung in Go (sehenswert):

https://www.youtube.com/watch?v=oV9rvDllKEg

Lektion 209 – WaitGroup – Warten wir mal, bis die da fertig sind

Eine WaitGroup wartet auf die Beendigung einer Anzahl von Go-Routinen. Die Haupt- Go-Routinen ruft die Methode `add()` auf, um die Anzahl der zu abzuwartenden Go-Routinen festzulegen. Dann wird jede der Go-Routinen ausgeführt und ruft `done()` auf, wenn sie endet. Gleichzeitig kann `wait()` dazu verwendet werden, um den Programmablauf zu blockieren, bis alle Goroutinen beendet sind. Das Schreiben von nebenläufigem Codes wird so supereinfach: Wir müssen nur ein "go" vor einen Funktions- oder Methodenaufruf setzen.

Aus dem Package runtime nutzen wir im Beispiel:

`runtime.GOOS`

`runtime.GOARCH`

`runtime.NumCPU()`

`runtime.NumGoroutine()`

und aus dem Package `sync`:

`sync.WaitGroup (als Dateityp)`

mit den Methoden:

`func (wg *WaitGroup) Add(delta int)`

`func (wg *WaitGroup) Done()`

`func (wg *WaitGroup) Wait()`

Anfangscode: https://go.dev/play/p/gttt4S8quEs

und unsere fertige WaitGroup: https://go.dev/play/p/Kms1TdoEYfJ

Lektion 210 – Method Sets reloaded - diesmal kennen sie keine Gnade

Das Method Set eines Typs bestimmt die Interfaces, die der Typ implementiert, und die Methoden, die von dem Receiver dieses Typs aufgerufen werden können.

Receiver ist Pointer, aber Wert ist keiner: https://go.dev/play/p/QWTlg-0vIJF **(Compilieren schlägt fehl!)**

Aber dieser Code funktioniert, beachten Sie den Unterschied durch den Einsatz von `c.fläche()`!

https://go.dev/play/p/XgAbSmSJHbH

Lektion 211 – Nebenläufigkeit - Ein Blick in die Dokumentation

Effektive Go: https://go.dev/doc/effective_go#concurrency

Go.dev: https://go.dev/ref/spec#Go_statements

Abildung aus e-book: https://livebook.manning.com/book/go-in-action/chapter-6/56

Zum Begriff „multiplexing“: https://de.wikipedia.org/wiki/Multiplexer

Lektion 212 – DIY Race Condition – Wer keine Arbeit hat, macht sich welche

Unsere Race-Condition: https://go.dev/play/p/yxGS6tm4Qx2

Im Texteditor kann man sich lokal auch eine `race-condition.go` erstellen und durch `go run race-condition.go` zur Ausführung bringen.

Die zusätzliche Angabe eines „Build Command“ -race zeigt uns durch den Compiler gefundene Race Conditions an: `go run -race race-condition.go`

Lektion 213 – Mutex

Was ist ein mutex: https://de.wikipedia.org/wiki/Mutex

Wo finden sich Mutex und deren Methoden in Go? https://pkg.go.dev/sync@go1.18.2#Mutex

Unsere Race-Condition durch Anwedung von Mutex entfernt: https://go.dev/play/p/Tjw_3QMV4gJ

Lokal findet `go run -race race-condition.go` auch keine Race Conditions mehr.

Lektion 214 – Package Atomic

Das Paket Atomic findet sich im Verzeichnis `sync` (ebenfalls ein Package): https://pkg.go.dev/sync/atomic#pkg-overview

Atomic bring eigene Methoden zur sicheren Manipulation und Lesen von Context für Go-Routinen mit: https://pkg.go.dev/sync/atomic#pkg-functions

Unsere Race-Condition durch Anwendung von von zwei Methoden aus dem Paket `sync/atomic` beseitigt: https://go.dev/play/p/keO4RPnNyH6

Bitte beachten Sie, dass die Ausgabe der Menge der laufenden Go-Routinen und des Counters naochmals angepasst wurde. Interessant ist zu sehen, dass durch Einsatz des Schedulers des Betrieibssystems, der Counter nicht zwingend (!) in der richtigen Reihenfolge hochgezählt wird. Beachten Sie die „C:“-Werte in der Reihe bei der Ausgabe.

Lektion 215 – Übung 1

Zusätzlich zur Hauptgoroutine zwei weitere Go-Routinen starten.

Jede zusätzliche Goroutine sollte etwas ausgeben.

Verwenden Sie WaitGroup, um sicherzustellen, dass jede Go-Routinen beendet werden kann, solange das Programm existiert.

Lektion 216 – Übung 1 - Beispiellösung

Beispiellösung: https://go.dev/play/p/M6cOmgwuOaF

Lektion 217 – Übung 2

Diese Übung soll das Verständnis von Method Sets vertiefen:

- erstellen Sie ein Struct vom Typ `Person`
- mit Hilfe eines Pointer Receivers weisen Sie eine Methode `speak` diesem Typ Person zu: `*Person`
- Erstellen Sie ein Interface vom Typ `Human` und fordern sie darin, dass ein Human die Methode `speak` implementieren muss, um als vom Typ `Human` zu gelten.

- Erstellen Sie die Funktion mit dem Identifier `saySomething`, die einen Wert vom Typ `Human` als Parameter entgegennimmt.
- Die Funktion soll die Methode `speak` aufrufen.

Stellen Sie im Code dar:

- Sie können einen Wert vom Typ `*Human` an `saySomething` übergeben
- Sie können nicht einen Wert vom Typ `Human` an `saySomething` übergeben
- Sie können problemlos `wertOfTypPerson.speak()` aufrufen!

Hinweis, falls Sie Hilfe benötigen: https://go.dev/play/p/UpIsJaflj_f

Lektion 218 – Übung 2 - Beispiellösung

Beispiellösung: https://go.dev/play/p/oYSVfPSOpr9

Lektion 219 – Übung 3

Erstellen Sie mit Hilfe von Go-Routinen ein Programm, das

- eine Variable, die den Wert des Zählers enthält, beinhaltet

und

- eine Reihe von Go-Routinen startet

Jede Goroutine soll den Zähler lesen, ihn in einer neuen Variablen speichern, die Prozessanforderung mit runtime.Gosched() beenden, die neue Variable erhöhen und zurück in die Zähler-Variable schreiben. Verwenden Sie WaitGroup, um auf das Ende aller Ihrer Go-Routinen zu warten.

Erzeugen Sie so eine und beweisen Sie das, indem Sie den Code mit dem „Build Flag“ -race compilieren und auf ihrem lokalen System zur Aufführung bringen.

Hinweis, falls Sie Hilfe benötigen: https://go.dev/play/p/FYGoflKQej

Lektion 220 – Übung 3 - Beispiellösung

Beispiellösung: https://go.dev/play/p/R3u8bOU2nwP

Lektion 221 – Übung 4

Benutzen Sie Methoden, die in Package `sync` für den Typ `Mutex` angeboten werden, um die Race Condition aus dem Code in Übung 3 zu umgehen.

Hinweis: Es macht Sinn, `runtime.Gosched()` zu entfernen. Warum?

Lektion 222 – Übung 4 - Beispiellösung

Beispiellösung: https://go.dev/play/p/0ChU2pktybx

Lektion 223 – Übung 5

Benutzen Sie Methoden, die in Package `sync/Atomic` angeboten werden, um die Race Condition aus dem Code in Übung 3 zu umgehen.

Lektion 224 – Übung 5 - Beispiellösung

Beispiellösung: https://go.dev/play/p/aJ0SqX5a71t

Lektion 225 – Übung 6

Erstellen Sie ein kleines Programm, das Ihr aktuelles OS und die CPU-Architektur auf der Konsole ausgibt.

Übernehmen Sie es lokal auf Ihren Rechner und führen Sie es aus mit:

```
go run
```

```
go build
```

Lektion 226 – Übung 6 - Beispiellösung

Beispiellösung: https://go.dev/play/p/6DGZUkaPPOj

Lektion 227 – Einführung und Erläuterungen zu Channels

https://go-proverbs.github.io/

Don't communicate by sharing memory, share memory by communicating. (Link führt zu Video, in dem Rob Pike die Proverbs erklärt - sehenswert)

https://go.dev/doc/effective_go#channels

https://pkg.go.dev/go/types#Chan

https://go.dev/ref/spec#Channel_types

Lektion 228 – Channels TL;DR; Channels block (die sind halt störrische Konstrukte!)

Channels block(ieren)!

Simples Beispiel, dass ein (unbuffered) Channel die weitere Programmausführung blockiert: https://go.dev/play/p/djgmRm6p_iN

Beispiel, dass ein (unbuffered) Channel, nur die eine nebenläufig ausgeführte Go-Routine aber nicht die weitere Programmausführung blockiert: https://go.dev/play/p/LbHMMqWlTbr

Beispiel, dass ein (buffered) Channel, nur die eine nebenläufig ausgeführte Go-Routine erst blockiert, wenn sein Buffer („Capacity“) überzogen wird: https://go.dev/play/p/8KchswQbrSH

Lektion 229 – Direktionale Channels – Geben Sie der Existenz Ihres Channels eine Richtung

Ausgangsbeispiel: https://go.dev/play/p/XgXhTeNr3av

Receive-only channel, d.h, von diesem Channel kann man nur Werte **empfangen**.
https://go.dev/play/p/anjvBwXuJSE

Send-Only channel, an disen Channel können wir nur Werte **senden**.
https://go.dev/play/p/5ej1GtZZ44I

Beispiel für Versuche, unidirektionalen Channel Werte aus eines anderen unidirektionalen Channels zuzuweisen: https://go.dev/play/p/sFS8t2x2HOe oder https://go.dev/play/p/i2fgzh2dAR- (**beide schlagen fehl**) und einem bidirektionalem Channel, Werte eines unidirektionalen Channels zuzuweisen: https://go.dev/play/p/L_CAVx3RJgf (**schlägt fehl**)

Die richtige Idee, aber falsche Syntax bei der Conversion: https://go.dev/play/p/TVclw8Rqf3S (**nicht richtig**)

Un schließlich die richtige **Conversion von Werten aus dem bidrektionalen Channel in Werte vom Typ des unidirektioneln Channels**: https://go.dev/play/p/0zxCrUxQpMF

Lektion 230 – Channels nutzen - eine Art Anwendungsbeispiel

Channel in einer Go-Routine nutzen, die einen unidirektionalen Channel als Typ für seinen Parameter entgegennimmt, die nebenläufig als „Sender“ (Dateneingabe an den Channel) zum Hauptprogramm abläuft: https://go.dev/play/p/JhxgpXMGDZw

Lektion 231 – Range & Close - Channel zu, Affe tot?

Channel in Funktion „ausgelagert“ mit Daten befüllen (send) und mit Range die Daten aus dem Channel abrufen. (Channel wird gleich nach dem Senden geschlossen. Daten bleiben aber im Channel erhalten! https://go.dev/play/p/OzDGMraUAv0

Hier werden die Daten statt über einen „Send-Only Channel“ Konstrukt in einer Funktion, gleich in eine Go-routine mit einer anonymen Funktion gegeben und der Channel nach dem Senden geschlossen. Die Daten selbst werden wie oben mit Range aus dem Channel abgerufen:
https://go.dev/play/p/PgXEj7RNruN

Lektion 232 – Select - Wählen Sie Ihren Kommunikationskanal

Wie funktioniert das Select statement? Ähnlich wie Switch, aber im Unterschied dazu werden verschiedene Komunikationssituationen für Channels in den Cases unterschieden.

Spezifikation: https://go.dev/ref/spec#Select_statements

Beispiel für Select: https://go.dev/play/p/7COzIJQ3IYd

Lektion 233 – , ok – Hey, das ist nicht komma okay!

Einschub vorab ausgehend vom letzten Stand unseres Beispiel: https://go.dev/play/p/7COzIJQ3IYd

Experiment zum Schließen der Channels: https://go.dev/play/p/Gjjbe9lA-3b

Korrigiert und mit Channel Ende Typ „chan bool“: https://go.dev/play/p/9HkKACNxYit

Korrigiert und mit Channel Ende Typ „chan bool“ und Slices, in denen wir Werte aus den Channels sammeln: https://go.dev/play/p/ET0pRGmYFK7 (Zur Verdeutlichung, dass Channels, wie die Nullen zustandekommen, wenn Channel nicht „zusammen“ geöffnet und geschlossen werden.) Spielen Sie mit der Anordnung der close()-Anweisungen und der Menge der in die Channel geschiebenen Daten.

Lektion 234 – Fan in - Channels zum Trichter aufgebaut

Fan In: Daten aus verschiedenen Channels, die von verschiedenen Go-Routinen versorgt werden, werden in einen Channel zusammengeführt.

Beispiel 1: https://go.dev/play/p/P_3x85nvTMb

Beispiel 2 (Rob Pike): https://go.dev/play/p/BvtrPEIH_fP

Quelle:

https://go.dev/blog/io2013-talk-concurrency (aufgearbeitet von Andrew Gerrand), Original Slides: https://go.dev/talks/2012/concurrency.slide#25 (Slide 25)

Lektion 235 – Fan out - Fliegt, meine Hübschen, fliegt, fliegt!

Fan Out: Eine wiederkehrende Aufgabe, wird auf mehrere nebenläufige Go-Routinen verteilt.

Beispiel einer Aufgabenverteilung gleicher Aufgaben auf nebenläufige Prozesse: https://go.dev/play/p/qzrbUgDQORP (zum Beispiel alle Videos in einem Ordner dekodieren)

Beispiel einer Aufgabenverteilung gleicher Aufgaben auf eine begrenzte Anzahl nebenläufiger Prozesse (Begrenzung des „Durchsatz“): https://go.dev/play/p/7VdjbNI-xux

Lektion 236 – Package Context - Wir geben Go-Routinen einen Kontext

Das Package „context“ definiert den Typ Context, der Deadlines, Abbruchsignale und andere anforderungsspezifische Werte über API-Grenzen hinweg und zwischen Prozessen überträgt.

- context.Background: https://go.dev/play/p/cByXyrxXUf
- context.WithCancel

  Verwerfen CancelFunc: https://go.dev/play/p/XOknf0aSpx

  Nutzen von CancelFunc :https://go.dev/play/p/UzQxxhn_fm
- Beispiel: https://go.dev/play/p/Lmbyn7bO7e

  func WithCancel(parent Context) (ctx Context, cancel CancelFunc):
  https://go.dev/play/p/wvGmvMzIMW

  cancelling goroutines mit deadline
  func WithDeadline(parent Context, deadline time.Time) (Context, CancelFunc)
  https://go.dev/play/p/Q6mVdQqYTt

  mit timeout:
  func WithTimeout(parent Context, timeout time.Duration) (Context, CancelFunc)
  https://go.dev/play/p/OuES9sP_yX

  mit value:
  func WithValue(parent Context, key, val interface{}) Context
  https://go.dev/play/p/8JDCGk1K4P

Weitere Informationsquellen:
https://pkg.go.dev/context
https://go.dev/blog/context
https://medium.com/@matryer/context-has-arrived-per-request-state-in-go-1-7-4d095be83bd8
https://peter.bourgon.org/blog/2016/07/11/context.html

Lektion 237 – Übung 1

Bringen Sie diesen Codeschnipsel zum Laufen: https://go.dev/play/p/-DpZPo8o5JQ

a) mit Hilfe eines „func literal“ also einer „anonymen“ selbst-aufrufenden Funktion

oder alternativ (!):

b) indem Sie einen Buffer im Channel einsetzen.

Lektion 238 – Übung 1 - Beispiellösung

Beispiellösungen:

a) https://go.dev/play/p/SHr3lpX4so

b) https://go.dev/play/p/Y0Hx6IZc3U

Lektion 239 – Übung 2

Bringen Sie diese Codeschnipsel zum Laufen:

a) https://go.dev/play/p/_DBRueImEq

b) https://go.dev/play/p/oB-p3KMiH6

Lektion 240 – Übung 2 - Beispiellösung

Beispiellösungen:

a) https://go.dev/play/p/BhhgKXOYAgA

b) https://go.dev/play/p/QHxrG8UEiuq

Lektion 241 – Übung 3

Starten Sie mit: https://go.dev/play/p/py-gC656Wjg und holen Sie die Werte mit Hilfe einer mit „range“ verwendenden Schleife aus dem Channel (schließt Ausgabe ein).

Lektion 242 – Übung 3 - Beispiellösung

Beispiellösung: https://go.dev/play/p/Xdtka4Xmvm2

Lektion 243 – Übung 4

Starten Sie mit: https://go.dev/play/p/YxHwstTc3Jc und holen Sie die Werte mit Hilfe einer mit „select“-Statement aus dem Channel (schließt Ausgabe ein).

Lektion 244 – Übung 4 - Beispiellösung

Beispiellösung: https://go.dev/play/p/FulKBY5JNj

Lektion 245 – Übung 5

Starten Sie mit: https://go.dev/play/p/7aWqxdYLgYJ und verwenden Sie zwei mal ein „,ok“ (Komma okay)-Statement (vor und nach dem close()), um zu zeigen, dass der Channel leer ist und kein Wert mehr aus dem Channel stammt.

Lektion 246 – Übung 5 - Beispiellösung

Beispiellösung: https://go.dev/play/p/qh2ywLB5OG

Lektion 247 – Übung 6

Schreiben Sie ein Programm, das 100 Werte in einen Channel schreibt (an einen Channel sendet) und anschließend alle Werte von diesem Channel empfängt und ausgibt.

Lektion 248 – Übung 6 - Beispiellösung

Beispiellösung: https://go.dev/play/p/QYM9NVIjAf-

Lektion 249 – Übung 7

Schreiben Sie ein Programm, das 10 Go-Routinen started und lassen Sie jede Go-Routine 10 Zahlen in einen Channel schreiben. Empfangen Sie alle 100 Werte aus dem Channel.

Lektion 250 – Übung 7 - Beispiellösung

Beispiellösung:

a) https://go.dev/play/p/QaC8983EqHU

oder

b) https://go.dev/play/p/A-uPci5uQ35 r

oder

c) https://go.dev/play/p/WqYnBC_CiKn

Lektion 251 – Übersicht: Notwendigkeit von Fehlerbehandlung verstehen

Fehler?
Wir machen einfach keine!

Falsch!

Fehler werden unterschätzt. In fehlerbehaftetem Code steckt mehr Infomation als in fehlerfreiem Code. Fehler haben einen Wert.

Syntax-Fehler werden weitestgehend abgefangen. Logikfehler, die aber syntaktisch erlaubt sind, sind zum Beispiel einen Wert zu übergeben, wo ein Pointer nötig wäre oder umgekehrt. Oft byzantinische Fehler, die man nicht sofort erkennen kann.
https://de.wikipedia.org/wiki/Byzantinischer_Fehler

Laufzeitfehler / Runtime Error wie „Teilung durch Null“ und andere zu erwartende Fehler, wie eine Datei ist nicht vorhanden, wenn wir sie beschreiben wollen, sind ganz anders. Eine besondere Klasse der Runtime Errors sind Exceptions. Fehler, die eigentlich nicht passieren sollten, aber wo der Compiler zur Laufzeit nicht feststellen konnte, dass sie auftreten könnten und auch nicht bewerten kann, ob die beabsichtigt sind oder nicht. Zugriff auf einen nicht reservierten Speicher zum Beispiel. Woher soll der Compiler wissen, ob dieser Zugriff von uns gewünscht ist oder nicht. Wenn da aber eine Anweisung steht, die unser Programm zerschießt, ist das eben eine Exception. Das kann aber auch ganz simpel, sein, wie „der Drucker hat kein Papier mehr“.

Beispiel mit Bezug zu „Div by zero“:

https://news.ycombinator.com/item?id=6233968

https://go.dev/play/p/aCraGRTaFi-

https://go.dev/play/p/HFzgX2VWx5t

In Go kennt keine Exceptions, weiterführend warum keine Exceptions:

Die Go-FAQ sagen:https://go.dev/doc/faq#exceptions

Quora sagt: https://www.quora.com/Why-does-Go-not-have-exceptions?q=why%20does%20go%20not%20have%20exce

Fehler sind Werte und haben Wert: https://blog.golang.org/errors-are-values

Fehler dort behandeln, wo sie potentiell auftreten. Und nicht dort sammeln, merken und irgendwo zentral sammeln. Wenn meine Kinder mit Lego spielen und ich nicht im Dunkeln auf einen treten will, sollte ich auch während die Kinder dort spielen, einen Hinweis geben, dass sie aufräumen und nicht die Gefahr aufschreiben, bis abends warten und die potentiellen Gefahren umschiffen.

Beschreibung:
https://go.dev/ref/spec#Errors

https://pkg.go.dev/errors

https://go.dev/doc/effective_go#errors

Proverbs: https://go-proverbs.github.io/

> **Errors are values.**
>
> **Don't just check errors, handle them gracefully.**

Lektion 252 – Auf Fehler prüfen meint prüfen und auch behandeln/abhandeln!

Wo immer möglich, sollte man auf Fehler prüfen, außer man begibt sich auf eine Art „Endlosschleife“. Die endgültige Entscheidung, wie tief Ihre Fehlerprüfung und Behandlung gehen soll, bleibt Ihnen überlassen, aber bei Funktionen wie z.B. `fmt.Println()` geht man davon aus, dass sie keinen Fehler ausgeben. Sonst müsste man bei der Ausgabe der Fehlers ja wieder auf Fehler püfen usw… Doch man kann das in go durchaus tun.

Beispiel 1: https://go.dev/play/p/N-w-Lbdr8Zp

Beispiel 2 (aus Package `fmt scan`, läuft nicht im Playground):
https://go.dev/play/p/6tNMWlBpwzd

Beispiel 3 (Datei schreiben, läuft nicht im Playground): https://go.dev/play/p/znDkCzv9mv-

Beispiel 4 (Datei lesen, läuft nicht im Playground): https://go.dev/play/p/oKQr-NKsYaY

Lektion 253 – Fehlerausgabe und in Logdateien schreiben

Für den Umgang mit Fehlermeldungen haben wir eine kleine Auswahl an Optionen zur direkten Ausgabe, Schreiben in Log-Dateien und Fehlermeldungen.

`fmt.Println()`

Ausgabe während des Programmablaufes auf Konsole (`stdout`)

Beispiel: https://go.dev/play/p/BFR5HUmIibK

`log.Println()`

Ausgabe während des Programmablaufes auf Konsole (default: stdout) oder in eine Datei.
Timestamp wird vorangestellt.

Beispiele:

https://go.dev/play/p/TOxT0DJJQgx (default stdout)

https://go.dev/play/p/8jpOgXLm86P (umgeleitet in Datei)

`log.Fatalln()`

Im Falle eines fatalen Fehlers, wird in die Logdatei geschrieben, aber der übergeordnete Codeblock direkt verlassen. os.Exit(1) bedeutet Programmabbruch!

Beispiel: https://go.dev/play/p/PtTRa0JSHK_Z

`log.Panicln()`

- verzögerte Functions laufen noch ab.
- `panic()`
- “recover” can aufgerufen werden (siehe nächste Lektion)

Beispiel: https://go.dev/play/p/PtTRa0JSHK_Z

Vorbreitung auf die nächste Lektion:

https://go.dev/ref/spec#Run_time_panics

https://go.dev/ref/spec#Handling_panics

Lektion 254 – Recovering - von Fehlern erholen

Recover funktioniert: https://go.dev/blog/defer-panic-and-recover

Beispiel defer: https://go.dev/play/p/HI4uG55ait

Beispiel für Umgang mit defer und panic! https://go.dev/play/p/OT_cnc_FwPt

Lektion 255 – Fehler mit Ansage und weitere Informationen

Wir können unseren Fehlern zusätzlich Informationen mitgeben.

- `errors.New()` und `fmt.Errorf()`
- `builtin.error`

> **“Error values in Go aren’t special, they are just values like any other, and so you have the entire language at your disposal.”** - Rob Pike

Beispiele:

`errors.New()`: https://go.dev/play/p/5ch0OagzqoV oder error in einer Variablen: https://go.dev/play/p/JDeNayjBIS8

`fmt.Errorf()`: https://go.dev/play/p/2tVGMkb77nP

`fmt.Sprintf`, um eine Fehlermeldung zu generieren und ein Struct, das einen Fehler beinhaltet: https://go.dev/play/p/PNubjVrt-ME

Lektion 256 – Übung 1

Starten Sie mit diesem Codeschnipsel: https://go.dev/play/p/vo4sREWnURK

Anstelle mit dem Underline/Unterlinie-Identifier die Fehlerinformation zu verwerfen, prüfen Sie auf Fehler im Rückgabewert und behandeln Sie den Fehler in angemessener Weise.

Lektion 257 – Übung 1 - Beispiellösung

Beispiellösung: https://go.dev/play/p/f8REMr_ICch

Lektion 258 – Übung 2

Starten Sie mit diesem Codeschnipsel: https://go.dev/play/p/0OXaX6NaNCQ

Erstellen Sie eine Fehlermeldung und nutzen Sie fmt.Errorf

Lektion 259 – Übung 2 - Beispiellösung

Beispiellösungen:

https://go.dev/play/p/vIHx8wlzA1F

Lektion 260 – Übung 2 – Beispiellösung - Ergänzung

Weitere Beispiellösungen:

https://go.dev/play/p/9sJUvxX3hRJ

https://go.dev/play/p/U5X25SQTXju

Lektion 261 – Übung 3

Erstellen sie einen Typ `eigenerFehler` mit unterliegendem struct, der das `builtin.error` Interface implementiert. Erstellen Sie eine Funktion `foo()`, die einen Wert vom Typ `error` als Parameter entgegennimmt. Danach erstellen Sie einen Wert vom Typ `eigenerFehler` und übergeben diesen an `foo()`.

Lektion 262 – Übung 3 - Beispiellösung

Beispiellösungen: https://go.dev/play/p/HyUHJ1FzK1F

Auch Legitim: https://go.dev/play/p/Rx2M8hbqp8X mit kleiner Erinnerung, was Conversion ist: https://go.dev/play/p/P90wq5dvJBv

Lektion 263 – Übung 4

Starten Sie mit diesem Codeschnipsel: https://go.dev/play/p/diHAKxQtVL8

Nutzen Sie das `wurzelschmerzenError` struct als Wert vom Typ `error`.

Wenn Sie mögen, nutzen Sie Lattitude "51.123 N" und Longitude "91.123 W" als weitere Werte.

Lektion 264 – Übung 4 - Beispiellösung

Beispiellösungen: https://go.dev/play/p/yJIsV_8acBf

Lektion 265 – Einführung und Übersicht

Software wird viel öfter gelesen, als geschrieben. Unvollständige, schlechte oder fehlende Dokumentation führt zu folgendem Mißstand bei der Softwareentwicklung.

Ein Entwickler erstellt eine Funktion und passt die aus diversen Gründen an. Es besteht dringender Bedarf an der Implementierung oder der Druck vom Team (auch gerade bei agiler Softwareentwicklung) ist so groß, dass die Dokumentation der eigenen Arbeit oft hinten angestellt oder gar komplett außer Acht gelassen wird. Später rächt sich das, wenn ein Nachfolger mit der gleichen oder ähnlichen Aufgabe konfrontiert, den Code lesen und verstehen muss. Die Zeit, die es braucht, um undokumentierten Code zu verstehen, überschreitet oft die Zeit, zu versuchen, ein kleines Modul einfach neu zu schreiben – natürlich wieder ohne Dokumentation. Das führt zu einer scheinbar endlosen Kette an undokumentiertem Code, den niemand mehr als Einzelperson nachvollziehen kann. Und DEN Programmierer als Einzelkäpfer gibt es ohnehin praktisch nicht mehr, stattdessen finden sich Programmierer eher bei Systemhäusern oder den großen Softwareschmieden oder eben Enterprises wie Google, Microsoft, Meta, Amazon, Netflix usw. Dort hat man den Wert und die Wichtigkeit einer vollständigen und einfachen Dokumentation schnell erkannt. Aus der „Sitte“ seinen Code hier und da mit guten Kommentaren zu unterfüttern wurde schnell eine Tugend.

Und Go macht aus der Tugend eine Kunst. In Go ist es praktisch schon die Funktionalität eingebaut, aus gut gepflegten Kommentaren, brauchbare Dokumentation zu extrahieren. Dazu gibt es nur einen Minumim an Aufwand zu erfüllen und wenige Voraussetzungen zu kennen und Vereinbarungen einzuhalten. Die Belohnung ist ein Dokumentationslevel, der es möglich macht, Code bis in die letzte kleine Funktion zu verstehen und diese Information jedem zugänglich zu machen.

godoc.org war einst eine Anlaufstelle, wo die Standard Library und Packages von Drittanbietern verwaltet und dokumentiert wurden. Und es gab golang.org wo die Standard Library alleine dokumenitiert war.

Go ist relativ jung und evetuell stoßen sie auf diese Domains noch. Sie sind aber zusammengeflossen in **go.dev**, bzw, **pkg.go.dev**. (auch der Playground war früher auf play.golang.org zu finden und liegt heute auf go.dev/play) und Links können so weiterin verwendet werden.

`go doc` ist ein Kommando, mit dem man die Dokumentation auf der Konsole lesen kann.

`godoc` dient ebenfalls dazu, mit dem man die Dokumentation auf der Konsole lesen und kann darüber hinaus die Dokumentation aber ansehnlich (als HTML) über einen selbst gestartenen Webserver lokal im Browser verfügbar machen.

Lektion 266 – Go doc - alles, was man so braucht, auf der Konsole

go doc gibt die Dokumentation aus für Package, Konstanten (const), Funktionen (func), Typen (type), Variablen (var) und Methoden (method)

go doc nimmt kein, ein oder zwei argumente entgegen.

kein Argument: Gibt die Dokumentation für das Paket im akutellen Verzeichnis aus

ein Argument als Go-syntax-artige Representant des Element, dessen Dokumentation man sehen will.

Beispiele (<sym> steht hier für “Identifier”)

- `go doc <pkg>`
- `go doc <sym>[.<method>]`
- `go doc [<pkg>.]<sym>[.<method>]`
- `go doc [<pkg>.][<sym>.]<method>`

Es gilt: Angezeigt wird das erste erfolgreich gefundene Element in dieser Liste. Wenn es ein <sym>, aber kein Paket gibt, wird das Paket im aktuellen Verzeichnis angezeigt. Wenn jedoch das

Argument mit einem Großbuchstaben beginnt, wird immer angenommen, dass es sich um ein

<sym> innerhalb des aktuellen Verzeichnisses handelt.

zwei Argumente

Erstes Argument muss ein vollständiger Paketpfad sein

Beispiel: `go doc <pkg> <sym>[.<method>]`

Lektion 267 – Godoc - Dokumentation ansehnlich

Godoc extrahiert und generiert Dokumentation für Go-Programme. Es unterstützte früher zwei Modi:

- mit -http Flag

Startet einen webserver und präsentiert die Dokumentation auf einer Website

Beispiel (einfach): `godoc` oder `godoc -http=:8080`

Startet einen Webserver erreichbar auf http://localhost:6060 oder auch http://127.0.0.1:8080

Beispiel (mit durchsuchbarem Index) `godoc -http=:8080 -index`

Beispiel für Nutzung mit aktiviertem Playground: `godoc -http=:8080 -play`
und Test mit Playground für die Codebeispiele (Examples):
http://localhost:8080/pkg/encoding/json/#example_Unmarshal

(VERALTET AIK, hier nur aus historischen Gründen)

- ohne -http Flag:

Kommandozeilen-orientierter Modus
Gibt eine Text-Dokumentation aus und beendet sich
-src Flag: godoc gibt das exportierte Interface eines Pakets in Go-Quellform aus oder die Implementierung einer bestimmten exportierten Sprache

Lektion 268 – pkg.go.dev (ehemals godoc.org)

pkg.go.dev (ehemals godoc.org)

Beispiel für einen einfachen Go-Code, den wir in einem Repository auf Github veröffentlichen können:

```
meaning
├── deepThought
│    └── deepThought.go https://go.dev/play/p/CXEd4Q0N9K8
├── LICENCE
├── main.go https://go.dev/play/p/KYLS_nlChDZ
└── README.md
```

Entweder legt man die komplette Struktur als Projekt in seinem Go-Pfad im Unterordner src an oder man erstellt einen symlink zu seinem Projektordner (unter Windows: https://www.howtogeek.com/howto/16226/complete-guide-to-symbolic-links-symlinks-on-windows-or-linux/), um die Daten für Go (und damit für godoc) verfügbar zu machen.

Symlinks unter Linux: `ln -s /Zieldatei/oderOrdener /Referenz/oderOrdner`

Mehr über Hardlinks/Softlinks in Deutsch auf:
https://www.ionos.de/digitalguide/server/konfiguration/linux-ln-befehl/

Dieses Projekt kann man nun auch auf github.com in einem eigenen Repository erreichbar machen.

Kopiert man die extern erreichbare URL (ohne führendes http://, bzw, https:// aber vielleicht von unseren Repository auf Github.com) von Go-Code hinter die URL https://pkg.go.dev/ erscheint die Dokumentation nach einem Refresh in den Ergebnissen der Suche.

Lektion 269 – Schreiben von Dokumentation

Dokumentation ist ein wichtiger Bestandteil, um Software zugänglich und wartbar zu machen. Sie sollte gut geschrieben und genau sein, aber sie muss auch einfach zu schreiben und zu pflegen sein. Idealerweise sollte sie an den Code selbst gekoppelt sein, so dass sich die Dokumentation

zusammen mit dem Code weiterentwickelt. Je einfacher es für die Programmierer ist, eine gute Dokumentation zu erstellen, desto besser für alle.

Hinweise: https://blog.golang.org/godoc-documenting-go-code

godoc analysiert den Go-Quellcode - einschließlich der Kommentare - und erstellt die Dokumentation als HTML oder reinen Text. Das Endergebnis ist eine Dokumentation, die eng mit dem dokumentierten Code verbunden ist. Über die Weboberfläche von godoc können Sie zum Beispiel mit einem Klick von der Dokumentation einer Funktion zu deren Implementierung navigieren.

Kommentare sind dann gute Kommentare, wenn man auch lesen möchte, wenn es godoc nicht gäbe.

Um Folgendes zu dokumentieren:

- Package,
- Konstanten (const),
- Funktionen (func),
- Typen (type),
- Variablen (var),
- Methoden (method)

Man schreibt einen Kommentar direkt vor die Deklaration, ohne eine Leerzeile zu lassen.

Man beginnt immer mit dem Namen des zu kommentierenden Elements.

Für Packages gilt:

- der erste Satz erscheint separat in der Package liste
- Hat man eine große Menge an Text, erstellt man besser die Dokumentation in einer Datei doc.go (Beispiel ist da das Package fmt)

Das Beste an dem minimalen Ansatz von godoc ist, wie einfach er zu verwenden ist. Der mit Abstand größte Teil der offiziellen Quellen, einschließlich der gesamten Standardbibliothek, und auch des allgemein verfügbaren Codes von Drittanbietern, folgt diesen Konventionen.

Lektion 270 – Übung 1

Erstellen Sie ein Package „dog“. Das Package sollte eine exportierte Funktion "Years" haben, die

Menschenjahre in Hundejahre umrechnet (1 Menschenjahr = 7 Hundejahre). Dokumentieren Sie den Code mit Kommentaren. Verwende diesen Code in ihrer Funktion main, um das Paket zu importieren: https://go.dev/play/p/5k71SfeL-YU

a) Führen Sie Ihr Programm aus und stellen Sie sicher, dass es funktioniert

b) Lassen Sie einen lokalen Server mit godoc laufen und sehen Sie sich Ihre Dokumentation an.

Lektion 270 – Übung 2

Veröffentlichen Sie den Code auf Github. Holen Sie sich Ihre Dokumentation zu pkg.go.dev und machen Sie einen Screenshot. Löschen Sie Ihren Code von github. Prüfen Sie auf pkg.go.dev und stellen Sie sicher, dass Ihr Code dort nicht mehr vorhanden ist.

Lektion 271 – Übung 1 & 2 – Beispiellösung

Beispiellösung: https://go.dev/play/p/DTVFLeBT1IT

Lektion 272 – Übung 3

Verwenden Sie `go doc` auf der Befehlszeile, um die Dokumentation für zu lesen für:

- fmt
- fmt Print
- strings
- strconv

Lektion 273 – Übung 3 – Beispiellösung

Lektion 274– Einführung und Übersicht über Tests und Benchmarks in Go

Einführung in Stichworten.

Tests erfordern

- eine Datei, deren Dateiname auf `_test.go` endet!
- in demselben Ordner/Package, wie der/das zu testende Element/Code (muss demselben Package) angehören.
- der Test läuft in einer Funktion mit einer Signatur mit der Struktur
  `func TestAbc(*testing.T)“`

Test ausführen mit

```
go test
```

Um das Ergebnis des Test zu behandeln nutzen wir `t.Error` um den Fehler zu signalisieren.

Üblicherweise in der Form: „Expected x, got y.“

Beispiel aus Video: https://go.dev/rplay/p/Qzpp5xtVBNy

Lektion 275 – Table Tests - Verhalten wie am Fließband testen

Beispiel aus Video: https://go.dev/play/p/AqJiYvkAdUI

Lektion 276 – Examples erlauben die Kombination von Dokumentation und Tests

Tests können sich in der Dokumentation als Examples wiederfinden.

Lokal können Sie schauen nach Start von: `godoc -http=:8080`

Weiterführende Informationen: https://go.dev/blog/examples

Beispiel aus Video: https://go.dev/play/p/luelx9aFO8-

Package mit testfiles und example auf pkg.go.dev:
https://pkg.go.dev/github.com/jagottsicher/meaning

Lektion 277 – Staticcheck: Schöner und einfacher

Wenn man Code nach bestimmten Kriterien „säubert“ und quasi von schlechtem Stil befreit, spricht man von „Linting“. Oft ist das eine undankbare Aufgabe, aber zum Glück gibt es dafür sogenannter Linter gibt. Golint war früher, wird aber nicht weiterentwickelt:
https://pkg.go.dev/golang.org/x/lint#section-readme

golint unterschied sich von `go fmt` in der Hinsicht, dass es über reine Formatierung wie Einrückungen hinaus, Ratschläge und Hinweise gibt, wie der Code verbessert werden sollte. Aber heute sollten Tools wie https://staticcheck.io/ (auf Github: https://github.com/dominikh/go-tools) stattdessen eingesetzt werden und viele IDE (auch VS Code) bringen viele Features für Linting bereits mit.

`go vet` geht noch einen Schritt weiter und verweist auf verdächtig aussehende Konstrukte und Strukturen, und hilft Code weiter zu vereinfachen.

Lektion 278 – Benchmarks/BET: Wir gehen mit schlechtem Beispiel voran

Unser Beispiel, von dem wir ein Benchmark erstellen wollen:

```
myGoBenchmark
├── LICENSE
├── main.go https://go.dev/play/p/es8FKhkWWkX
├── myConcat
│   ├── myConcat.go https://go.dev/play/p/QdclGwwcUn8
│   └── myConcat_test.go https://go.dev/play/p/QYU-Cks5lLp
└── README.md
```

Lektion 279 – Benchmarks/BET: Lasst die Spiele beginnen!

Vergleich unserer einfachen Concat-Implementierung mit der Funktion strings.Join und anschließendem Performance-Vergleich.

```
myGoBenchmark
├── LICENSE
├── main.go https://go.dev/play/p/YJgBI_YURTl
├── myConcat
│    ├── myConcat.go https://go.dev/play/p/PrfWorZowOF
│    └── myConcat_test.go https://go.dev/play/p/I3g3jeu9okV
└── README.md
```

Lektion 280 – Über die Abdeckung von Go Code in Tests

Coverage beim Programmieren meint wieviel unseres Codes von Tests abgedeckt ist.

Mit dem “-cover” Flag wird eine Testabdeckungs-Anylyse gestartet.

Mit “-coverprofile <irgendeinName>” wird eine Anylyse in eine Datei geschrieben.

Beispiele:

```
go test -cover
```

```
go test -coverprofile analyse.txt
```

```
go tool cover -html=analyse.txt
```

Weitere Infos: `go tool cover -h`

Lektion 281 – Zusammenfassung BET

Denken Sie an BET:

- Benchmark
- Example
- Test

Testfile erstellen, der auf _test.go endet und package „testing“ importiert:

```
func BenchmarkIhrIdentifier(b *testing.B)
```

Nützlich:

`b.ResetTimer()` (den Timer für die Benchmarks (zurück-)setzen)

`b.N` (Anzahl der Benchmarkläufe, für eine statistisch relevantes Mittel)

```
func ExampleIhrIdentifier()
```

```
// Output:
// Erwarteten Output hier kopieren!
func TestIhrIdentifiert(t *testing.T)
t.Error("Expected:", x, "got:", y)
```

Befehle:

```
godoc -http=:8080


go test
go test -bench .
go test -cover .
go test -coverprofile dateiname
go tool cover -html=dateiname
```

Lektion 282 – Übung 1

Gegeben sei ein Go-Projekt mit folgender Struktur:

```
myGoBETuebung
├── main.go https://go.dev/play/p/YdLyvzrhsSf
└── mathehelfer
    └── mathehelfer.go https://go.dev/play/p/yBOBtLUf1rm
```

Sie finden ein öffentliches Repository auf github.com:
https://github.com/jagottsicher/myGoBETuebung

Hinweis: Beachten Sie, dass sich der Ordner `mathehelfer` in Ihrem Pfad befinden muss.

a)
Führen Sie `main.go` aus und fügen Sie einen geeigneten Testfile hinzu für das Package `mathehelfer`.

b)
Fügen Sie einen Benchmark für die Funktion `DieSummeVon` im Package `mathehelfer` hinzu.

Führen Sie im Ordner des Package `mathehelfer` aus:

```
go test
go test -bench .
```

c)
Implemenieren Sie folgenden Code in Package `mathehelfer`:

```
func zweiGanzzahlenAddieren(a, b int) int {
     return a + b
```

```
}

func EineAndereArtSumme(eingabeWerte ...int) int {

	var summe int
	summe = eingabeWerte[0]

	for _, v := range eingabeWerte[1:] {
		summe = zweiGanzzahlenAddieren(summe, v)
	}
	return summe
}
```

und entfernen Sie die Auskommentierung in Zeile 13 in main.go im übergeordneten Ordner.

Führen Sie main.go aus.

d)
Führen Sie im Ordner des Package `mathehelfer` aus:

```
go test
go test -bench .
```

Fügen Sie einen weiteren Benchmark, aber diesmal für die Funktion `EineAndereArtSumme` im Package `mathehelfer` hinzu.

Führen Sie im Ordner des Package `mathehelfer` aus:

```
go test
go test -bench .
```

e)
Versehen Sie Datei `mathehelfer.go` mit Dokumentation in Form von Kommentaren für

- das Package `mathehelfer`
- Die Funktion `DieSummeVon`
- Die Funktion `EineAndereArtSumme`

Starten Sie

```
godoc -http=:8080
```

und rufen Sie http://localhost:8080 im Browser auf und finden Sie das Package `mathehelfer`

Stoppen Sie den „godoc-webserver“

f)
Fügen Sie jeweils ein Example für die Aufrufe der Funktionen `DieSummeVon` und `EineAndereArtSumme` hinzu. Die Werte 1,2 und 3 sollen als Argumente übergeben werden und die Ausgabe soll „Summe: 6“ lauten.

Führen Sie `go test` aus und sorgen Sie dafür, dass die Examples und der Test durchlaufen.

Starten Sie

```
godoc -http=:8080
```

und rufen Sie http://localhost:8080 im Browser auf und finden Sie die beiden Examples im Package `mathehelfer`

Erfreuen Sie sich an den Beispielen in der Dokumentation und stoppen Sie den „godoc-webserver“.

g)
Fügen Sie jeweils einen Test für die Aufrufe der Funktionen `DieSummeVon` und `EineAndereArtSumme` hinzu.

Testen Sie die Werte 1,2 und 3 als Argumente gegen das Ergebnis 6 und sorgen Sie für eine ordentliche Ausgabe im Falle von „FAIL“ in der Art „Expected: x, got y“ o.ä.

Führen Sie `go test` aus und sorgen Sie dafür, dass die Test durchlaufen.

h)
Führen Sie im Ordner des Package `mathehelfer` aus:

```
go test -cover .
```

```
go test -coverprofile dateiname
```

```
go tool cover -html=dateiname
```

Fügen Sie jeweils einen weiteren Test für die Aufrufe der Funktionen `DieSummeVon` und `EineAndereArtSumme` hinzu.

Diesmal testen Sie die Werte entsprechend nachfolgender Tabelle:

```
Argumente      Ergebnis
-----------------------------------------
1, 2, 3, 4, 5, 6         21
1, 2, 4, 8, 16           31
1, -2, 3, -4, 5          3
-5, 1, 1, 1, 1, 1        0
6, 5, 4, 3, 2, 1         21
```

Sorgen Sie für eine ordentliche Ausgabe im Falle von „FAIL“ in der Art „Expected: x, got y“ o.ä.

Führen Sie `go test` aus und sorgen Sie dafür, dass alle Tests durchlaufen.

Führen Sie im Ordner des Package `mathehelfer` aus:

```
go test
```

```
go test -bench .
```

```
go test -cover .
```

```
go test -coverprofile dateiname
```

```
go tool cover -html=dateiname
```

i)
Freuen Sie sich nochmal über eine gelungene Lösung für BET (Benchmark, Example, Test)

Lektion 283 - 291 – Übung 1 a) - i) Beispiellösung

Sie finden den kompletten Code der Beispiellösung auf Github:
https://github.com/jagottsicher/myGoBETuebung-loesung

Lektion 292 – Package Manager und Anhängigkeiten (Dependencies)

https://research.swtch.com/deps

PDF ist verfügbar hier: https://research.swtch.com/deps.pdf

Lektion 293 – Wie man Go Modules benutzt - allgemeiner Hinweise

> **"… Google's internal source code system, which treats software dependencies as a first-class concept, …"**
> *https://research.swtch.com/deps*

https://en.wikipedia.org/wiki/First-class_citizen

Quelle, an der wir uns entlanghangeln: https://go.dev/blog/using-go-modules und weiterführende Informationen in den nachfolgenden Teilen dieser Serie von Blogbeiträgen. Die Serie bietet einen umfassenden Einstieg in die Verwaltung von Abhängigkeiten mit Go Modules.

Lektion 294 – Selbst ein Go Modul erstellen

https://go.dev/blog/using-go-modules:

`go mod init` erstellt ein neues Modul und initialisiert die `go.mod` -Datei, die das Modul beschreibt.

Lektion 295 – Anhängigkeiten einem Go Modul hinzufügen

https://go.dev/blog/using-go-modules:

`go get` ändert die erforderliche Version einer Anhängigkeit und/oder fügt eine Abhängigkeit hinzu.

`go build`, `go test` fügt der `go.mod` nach Bedarf neue Anhängigkeiten zu.

`go list -m all` gibt alle aktuellen Anhängigkeiten des aktuellen Moduls aus.

Lektion 296 – Anhängigkeiten updaten/erfüllen/downgraden

https://go.dev/blog/using-go-modules:

`go get domäne/ordner`

`go get domäne/ordner@versionstag`

ändert die erforderliche Version einer Anhängigkeit und/oder fügt eine Abhängigkeit hinzu.

`go list -m -versions domäne/ordner` gibt eine Liste aller verfügbaren Versionen eines Moduls aus.

`go mod tidy` gleicht den Code unseres Go-Moduls mit den Angaben den bis dato erforderichen Anhängigkeiten aus `go.mod` ab und entfernt nicht mehr benötigte Anhängigkeiten und Module, fügt neue ggfls. hinzu.

Lektion 297 – Übung 1

Die finden den aktuellen Code aus den vorherigen Lektionen unter https://github.com/jagottsicher/myGoModulesBeispiel.

Mit `git clone git@github.com:jagottsicher/myGoModulesBeispiel.git` können Sie ihn sich herunterladen.

a)
Führen Sie in dem Projekt ein Upgrade nach einem großen Versionssprung aus, wie in https://go.dev/blog/using-go-modules#upgrading-a-dependency-to-a-new-major-version beschrieben.

b)
Entfernen Sie nicht mehr genutzte Anhängigkeiten wie in https://go.dev/blog/using-go-modules#removing-unused-dependencies beschrieben.

Nutzen Sie `go mod tidy`.

Lektion 298 – Übung 1 a), b) Beispiellösung

Eine Beispiellösung: https://github.com/jagottsicher/myGoModulesBeispielLösung

Lektion 299– Sie haben's geschafft - feiern Sie sich!

Lektion 300 – Hinter dem Horizont geht's weiter ...