IN DIESEM KAPITEL

- Vorteile von Go
- Go auf Ihrem Computer installieren
- Mit einer IDE arbeiten
- Go-Programme schreiben und ihre Funktionsweise verstehen
- Go im Vergleich mit anderen Programmiersprachen

Kapitel 1
Auf die Plätze, fertig, Go!

Go ist eine quelloffene Programmiersprache, die von Google erstmals im Jahr 2009 vorgestellt wurde und inzwischen immer mehr Fans findet. Mit dieser Sprache können Sie unterschiedlichste Aufgaben bewältigen und vor allem schnelle, skalierbare Anwendungen entwickeln.

Hinter Go stecken einige ziemlich clevere Köpfe, allen voran: Ken Thompson (Designer und Erfinder von Unix und C), Rob Pike (Co-Entwickler des UTF-8- und Unix-Formats) und Robert Griesemer (ein Google-Informatiker). Wenn Sie sich für technische Details interessieren und des Englischen mächtig sind, können Sie im Artikel »Go at Google: Language Design in the Service of Software Engineering« (`https://talks.golang.org/2012/splash.article`) nachlesen, welche Problemstellungen ursprünglich mit Go bei Google angegangen werden sollten.

In diesem Kapitel erfahren Sie, warum Go Ihrer Karriere zuträglich ist, wo diese Sprache überall eingesetzt werden kann und welche ersten Schritte Sie unternehmen müssen, um selbst mit Go zu programmieren.

Da die Website von Go unter der Adresse `https://golang.org` zu erreichen ist, hat sich auch die Bezeichnung *Golang* für diese Programmiersprache etabliert. Der offizielle Name ist jedoch Go und wird demzufolge in diesem Buch beibehalten.

Mit Go zum beruflichen Erfolg

Programmiersprachen gibt es heute viele, doch Go sticht aus der breiten Masse hervor. Dies hat mehrere Gründe:

- ✔ **Go lässt sich leicht erlernen.** Durch seine Syntax ist Go eine gut lesbare Sprache. Die objektorientierte Programmierung wird nicht unterstützt, sodass Sie sich keine Gedanken um Klassen und Vererbung und all die komplexen Merkmale dieses Programmierparadigmas machen müssen.

Bei der objektorientierten Programmierung (OOP) dreht sich alles um *Datenobjekte*. Es geht also primär nicht um Funktionen oder Logik, sondern um Daten. Ein wesentliches Konzept bei der OOP ist die *Klasse* (quasi eine Art Vorlage für Datenobjekte). Angenommen, Sie möchten Ihre Anwendung mit mehreren Schaltflächen versehen. Damit Sie den zugehörigen Programmcode nicht wiederholt für jede einzelne Schaltfläche schreiben müssen, können Sie eine Klasse für eine generische Schaltfläche definieren und diese dann zum Erstellen Ihrer gewünschten Schaltflächen nutzen. Dabei hat jede Schaltfläche ihre eigenen *Attribute* (Merkmale). Mithilfe von *Vererbung* können Sie bei der OOP mehrere *Subklassen* aus der generischen Schaltflächenklasse erzeugen, um unterschiedliche Arten von Schaltflächen zu generieren, beispielsweise Schaltflächen in Rechteckform oder mit abgerundeten Ecken und so weiter.

- ✔ **Go hat eine geringere Funktionsauswahl als andere Programmiersprachen.** Bei Go müssen Sie nicht erst überlegen, auf welche Weise Sie ein bestimmtes Problem am besten lösen – es gibt immer nur eine korrekte Vorgehensweise. Dadurch bleibt Ihre Codebasis viel übersichtlicher.
- ✔ **Go glänzt bei nebenläufiger Programmierung.** Durch sogenannte *Goroutinen* ist es äußerst einfach, mehrere Funktionen parallel auszuführen.

Go bietet derzeit keine Unterstützung für *generische Datentypen* (bei denen der tatsächliche Datentyp erst bei Verwendung angegeben werden muss), doch dies könnte sich künftig ändern, da die Sprache weiterentwickelt wird.

All das hat Sie noch nicht überzeugt? Dann habe ich ein weiteres Argument, das Sie motivieren könnte, die Programmierung mit Go zu lernen: Stack Overflow, eine Internetplattform mit Fragen und Antworten rund um das Thema Programmierung, führt jährliche Gehaltsumfragen unter Softwareentwicklern durch. Ergebnissen aus dem Jahr 2021 zufolge (siehe *Stack Overflow Developer Survey 2021*, `https://insights.stackoverflow.com/survey/2021`) landen Go-Entwickler unter den ersten zehn.

Obwohl es Go schon eine ganze Weile gibt (nämlich seit 2009), findet die Sprache erst jetzt immer mehr Zustimmung in der Softwareentwicklungsbranche, vor allem dank Cloud-Computing und Microservices. Mittlerweile kommt Go bei zahlreichen großen Unternehmen wie Dailymotion, Dropbox, Google und Uber zum Einsatz.

Die Anwendungsszenarien sind vielfältig:

- ✔ **Cloud-Services:** Mit Go und der Google Cloud Platform (GCP) können Sie skalierbare Apps erstellen.
- ✔ **Netzwerkanwendungen:** Durch die Unterstützung für Goroutinen lassen sich mit Go verteilte Serveranwendungen und Programmierschnittstellen (APIs) entwickeln.
- ✔ **Webservices:** Auch skalierbare und effiziente Webservices sind mit Go kein Problem.
- ✔ **Befehlszeilenbasierte Anwendungen:** Da Go auf mehreren Plattformen ausführbar ist, können Sie dieselbe Codebasis gezielt für verschiedene Plattformen kompilieren (zum Beispiel für macOS und Windows).

Installieren von Go auf Ihrem Computer

Bestimmt sind Sie neugierig, wie Sie nun mit Go etwas auf Ihrem Computer programmieren können. Legen wir also los!

Als Erstes müssen Sie Go installieren. Am einfachsten geht dies über die Installationsseite auf der offiziellen Go-Website: `https://golang.org/doc/install`. Die Website erkennt automatisch Ihr gerade genutztes Betriebssystem und bietet Ihnen die passende Installationsdatei zum Herunterladen an (siehe Abbildung 1.1).

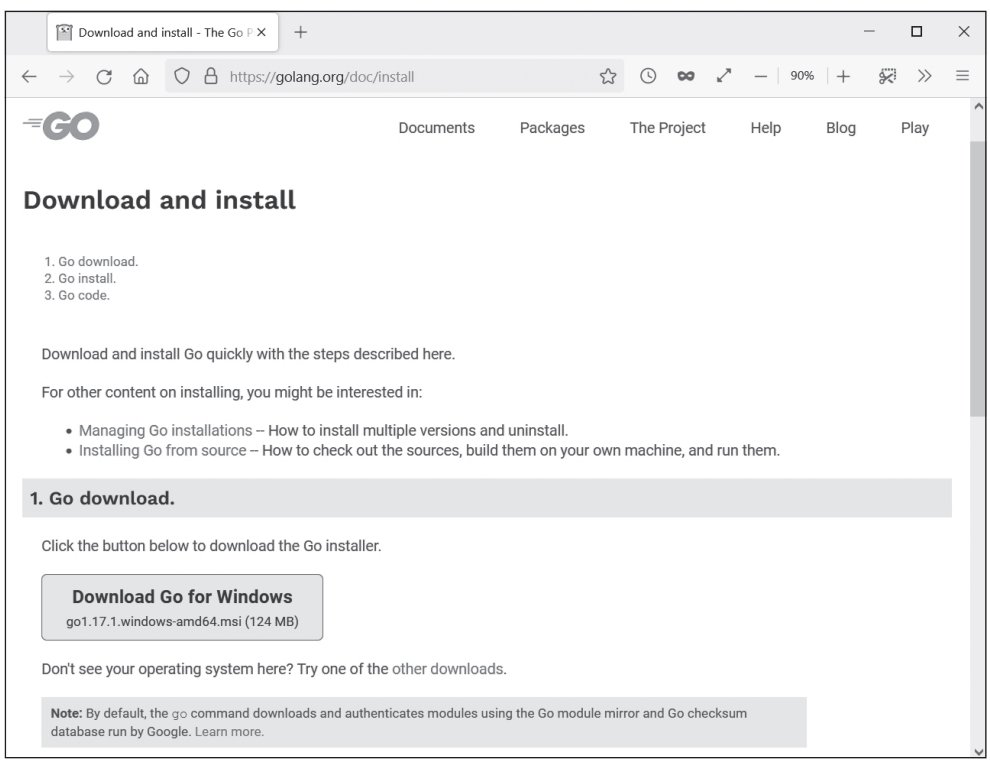

Abbildung 1.1: Die Go-Installationsdatei steht zum Herunterladen bereit.

Die Programmbeispiele in diesem Buch wurden mit Version 1.15 von Go geschrieben und getestet. Wenn Sie dieses Buch in den Händen halten, gibt es aber bestimmt schon eine neuere Go-Version. Damit Sie trotzdem alle Beispiele problemlos ausprobieren können, sollten Sie sich dieselbe Version herunterladen, die ich auch genutzt habe. Diese finden Sie hier:

- ✔ **macOS:** `https://golang.org/dl/go1.15.8.darwin-amd64.pkg`
- ✔ **Windows:** `https://golang.org/dl/go1.15.8.windows-amd64.msi`

Wenn Sie alle verfügbaren Installationsdateien für die unterstützten Plattformen (Linux, macOS und Windows) oder sogar den Quellcode von Go sehen möchten, öffnen Sie folgende Seite in Ihrem Browser: `https://golang.org/dl/`.

Nachdem Sie die Go-Installationsdatei heruntergeladen haben, führen Sie sie per Doppelklick aus, um den Installationsvorgang zu starten. Es öffnet sich ein Dialogfenster, in dem Sie der Endbenutzer-Lizenzvereinbarung zustimmen müssen und dann den gewünschten Speicherort für die Go-Dateien angeben können. Unter Windows wird standardmäßig der Ordner `Programme` beziehungsweise `Programme (x86)` vorgeschlagen. Damit Sie die Beispiele aus diesem Buch leicht nachvollziehen können, ändern Sie diese Einstellung und speichern Sie die Dateien stattdessen unter `C:\Go`. Alle anderen Standardeinstellungen können Sie beibehalten. Klicken Sie zum Schluss auf die Schaltfläche Install. Go wird nun innerhalb weniger Sekunden installiert. Nach Abschluss des Vorgangs erscheint ein Bestätigungsfenster (siehe Abbildung 1.2).

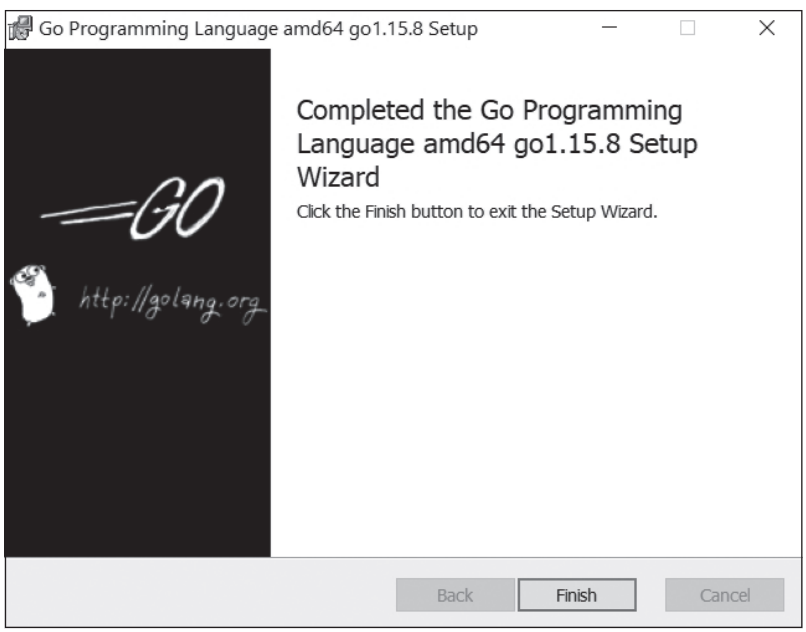

Abbildung 1.2: Der Go-Installationsvorgang ist abgeschlossen.

In den folgenden Abschnitten zeige ich Ihnen, wie Sie unter macOS und Windows überprüfen können, ob Go korrekt installiert wurde.

macOS

Unter macOS wird die Go-Distribution standardmäßig im Verzeichnis `/usr/local/go` installiert. Außerdem fügt das Installationsprogramm automatisch das Verzeichnis `/usr/local/go/bin` zur `PATH`-Umgebungsvariablen hinzu. Dies lässt sich leicht überprüfen. Öffnen Sie dazu die Terminal-App (die Sie im Ordner `Programme/Dienstprogramme` finden) und geben Sie folgenden Befehl ein:

```
$ echo $PATH
```

Sie sehen nun eine Ausgabe ähnlich der folgenden, nur dass statt `weimenglee` Ihr eigener Benutzername erscheint (der neu hinzugefügte Go-Pfad ist fett hervorgehoben):

```
/Users/weimenglee/opt/anaconda3/bin:/Volumes/SSD/opt/anaco
nda3/condabin:/Users/weimenglee/flutter/bin:/Users/weimeng
lee/go/bin:/Users/weimenglee/.nvm/versions/node/v9.2.0/bin
:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/g
o/bin:/usr/local/share/dotnet:~/.dotnet/tools:/Library/App
le/usr/bin:/Library/Frameworks/Mono.framework/Versions/Cur
rent/Commands
```

Starten Sie die Terminal-App neu, nachdem Sie Go installiert haben, damit die geänderten Einstellungen korrekt übernommen werden.

Um zu testen, ob bei der Installation alles geklappt hat, geben Sie folgenden Befehl im Terminal-Fenster ein:

```
$ go version
```

Ihnen sollte nun die auf Ihrem System installierte Go-Version angezeigt werden:

```
go version go1.11.5 darwin/amd64
```

Windows

Unter Windows wird die Go-Distribution im Verzeichnis `C:\Go` installiert (vorausgesetzt, Sie haben die Standardeinstellung geändert, wie zuvor beschrieben). Außerdem fügt das Installationsprogramm automatisch das Verzeichnis `Go\bin` zur `PATH`-Umgebungsvariablen hinzu. Auch dies lässt sich leicht überprüfen. Öffnen Sie dazu die Eingabeaufforderung (indem Sie zum Beispiel die Tastenkombination ⊞ + R drücken und dann im Dialogfenster `cmd` eingeben) und geben Sie folgenden Befehl ein:

```
C:\Users\Wei-Meng Lee>path
```

Sie sehen nun eine Ausgabe ähnlich der folgenden, nur dass statt `weimenglee` Ihr eigener Benutzername erscheint (der neu hinzugefügte Go-Pfad ist fett hervorgehoben):

```
PATH=C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\
Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WIND
OWS\System32\OpenSSH\;C:\Program Files\dotnet\;C:\Program
Files\Microsoft SQL Server\130\Tools\Binn\;C:\Go\bin;
C:\Program Files\Git\cmd;C:\Program Files\Graphviz
2.44.1\bin;C:\Program Files\CMake\bin;C:\Program
Files\Docker\Docker\resources\bin;C:\ProgramData\DockerDes
ktop\version-bin;C:\Program Files\MySQL\MySQL Shell
8.0\bin\;C:\Users\Wei-Meng Lee\AppData\Local\
Microsoft\WindowsApps;;C:\Users\Wei-Meng Lee\
AppData\Local\Programs\Microsoft VS Code\bin;C:\Users\Wei-
Meng Lee\.dotnet\tools;C:\Users\Wei-Meng Lee\go\bin
```

Starten Sie die Eingabeaufforderung neu, nachdem Sie Go installiert haben, damit die geänderten Einstellungen korrekt übernommen werden.

Um zu testen, ob bei der Installation alles geklappt hat, geben Sie folgenden Befehl in der Eingabeaufforderung ein:

```
C:\Users\Wei-Meng Lee>go version
```

Ihnen sollte nun die auf Ihrem System installierte Go-Version angezeigt werden:

```
go version go1.15.2 windows/amd64
```

Verwenden einer integrierten Entwicklungsumgebung (IDE) mit Go

Um Anwendungen mit Go zu entwickeln, benötigen Sie eigentlich nur einen Texteditor (zum Beispiel TextEdit unter macOS oder den klassischen Editor unter Windows). Die Profis in der Softwareentwicklung bevorzugen aber häufig integrierte Entwicklungsumgebungen (*Integrated Development Environments,* IDEs), mit denen sich der Programmcode besser organisieren und auf Fehler überprüfen lässt. Mit Go harmonieren zum Beispiel folgende IDEs:

- **Visual Studio Code (`https://code.visualstudio.com`):** Visual Studio Code aus dem Hause Microsoft ist eine IDE par excellence (und mein persönlicher Favorit). Das Tool umfasst alle üblichen Editorfunktionen und unterstützt fast jede erdenkliche Programmiersprache. Besonders nützlich ist die IntelliSense-Funktion, die Ihre Programmbefehle automatisch vervollständigt, während Sie sie eingeben. Visual Studio Code bietet außerdem einen Debugger, eine interaktive Konsole sowie integrierte Git-Unterstützung. Und das Beste: Visual Studio Code ist kostenlos erhältlich und erfreut sich unter Go-Entwicklern hoher Beliebtheit, sodass es zahlreiche Plug-ins gibt, mit denen Sie die Funktionalität sogar noch erweitern können.

- **GoLand (`www.jetbrains.com/de-de/go/`):** GoLand ist eine plattformübergreifende IDE des multinationalen Softwareanbieters JetBrains. Unter anderem bietet diese IDE Unterstützung bei der Befehlseingabe, einen Debugger und einen integrierten Terminal. GoLand ist ein kommerzielles Produkt, kann aber als Testversion 30 Tage lang kostenlos genutzt werden.

- **Go Playground (`https://play.golang.org`):** Beim Go Playground handelt es sich eigentlich nicht um eine IDE, sondern um einen Webservice, der über die Server der offiziellen Go-Website läuft und – wie der Name schon sagt – als »Spielplatz« dient, auf dem Sie kurze Go-Programmzeilen im Browser testen können. Das eingegebene Go-Programm wird kompiliert, verlinkt und innerhalb der Testumgebung ausgeführt, sodass Sie sehen, ob die Ausgabe des Programms Ihren Erwartungen entspricht.

Für die Go-Programmbeispiele und Erklärungen in diesem Buch nutze ich Visual Studio Code. Diese IDE steht unter `https://code.visualstudio.com/download` zum Herunterladen bereit.

Nachdem Sie Visual Studio Code heruntergeladen und installiert haben, starten Sie die Anwendung. Es öffnet sich ein Begrüßungsfenster, in dem standardmäßig alles auf Englisch angezeigt wird. Sofern Sie ein deutsches Betriebssystem nutzen, erscheint aber in der unteren rechten Ecke automatisch ein Hinweis auf das deutsche Sprachpaket. Um dies zu installieren, klicken Sie auf die Schaltfläche Installieren und neu starten (siehe Abbildung 1.3).

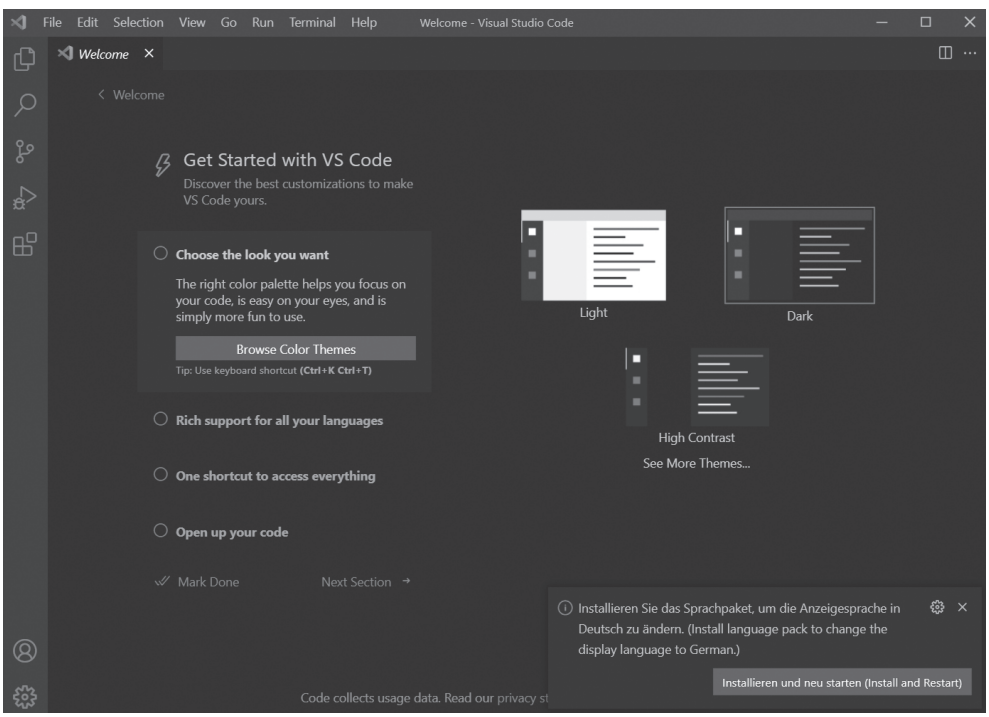

Abbildung 1.3: Visual Studio Code wird zum ersten Mal gestartet.

Das Sprachpaket wird automatisch installiert und Visual Studio Code neu gestartet. Bei Bedarf können Sie nun auch die Darstellungsoptionen anpassen. Im dunklen Standarddesign präsentiert sich Visual Studio Code wie in Abbildung 1.4 zu sehen.

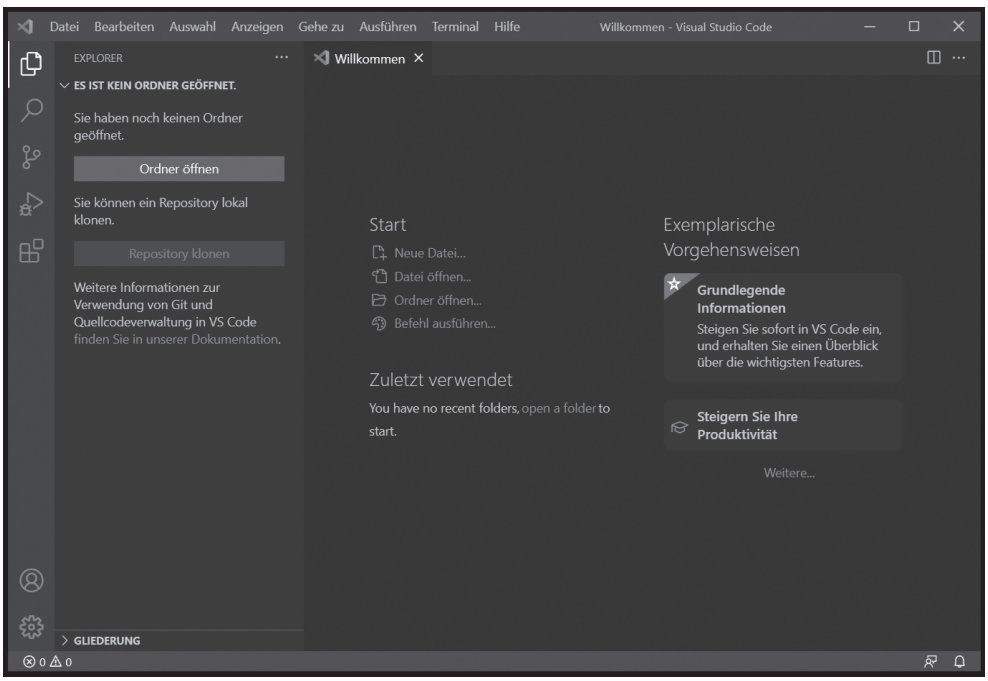

Abbildung 1.4: Visual Studio Code ist einsatzbereit.

Damit Visual Studio Code die Syntax von Go erkennt und verarbeiten kann, müssen Sie eine Erweiterung installieren. Führen Sie dazu folgende Schritte aus:

1. **Klicken Sie in Visual Studio Code in der Aktivitätsleiste am linken Rand auf das Symbol Erweiterungen (siehe Abbildung 1.5).**

Abbildung 1.5: Das Symbol Erweiterungen ganz unten in der Aktivitätsleiste

2. **Geben Sie in der Seitenleiste im Suchfeld das Stichwort `Go` ein.**

   In den Suchergebnissen sehen Sie diverse verfügbare Go-Erweiterungen (siehe Abbildung 1.6).

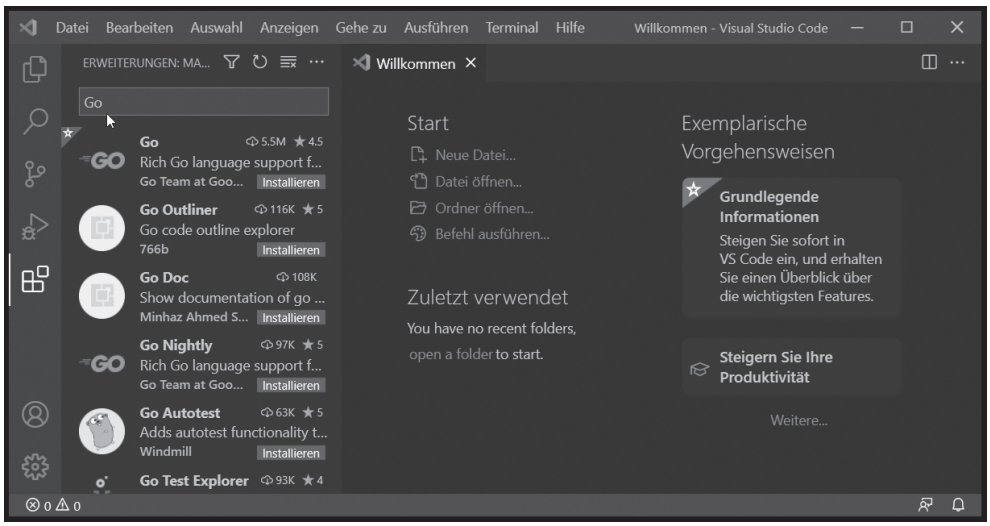

Abbildung 1.6: Für Visual Studio Code gibt es zahlreiche Go-Erweiterungen.

3. **Klicken Sie neben der ersten Erweiterung (»Rich Go language support for Visual Studio Code«) auf die Schaltfläche Installieren.**

   Das war's schon! Nun können Sie mit dem Programmieren loslegen.

Ihr erstes Go-Programm

Um Ihr erstes eigenes Programm in Go zu schreiben, erstellen Sie zunächst in Visual Studio Code eine neue Datei. Klicken Sie dazu in der Menüleiste auf Datei | Neue Datei. Geben Sie dann folgende Befehle ein (siehe Abbildung 1.7):

```
package main

import "fmt"

func main() {
    fmt.Println("Hallo Welt!")
}
```

Haben Sie alles eingetippt, drücken Sie cmd ⌘ + S (macOS) oder Strg + S (Windows), um diese Datei zu speichern. Wählen Sie als Dateinamen `main.go`. Wenn Sie zum ersten Mal ein Go-Programm in Visual Studio Code schreiben, öffnet sich unter Umständen ein Hinweisfenster, in dem Sie zusätzliche Plug-ins für Go herunterladen können. Ich empfehle Ihnen, die vorgeschlagenen Plug-ins alle zu installieren.

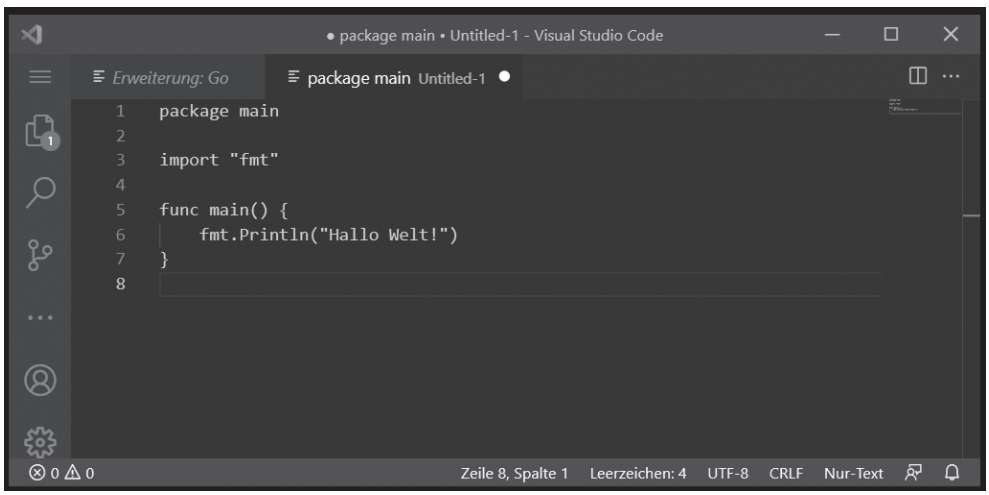

Abbildung 1.7: Zur Einführung in die Programmierung mit Go schreiben Sie ein typisches »Hallo Welt«-Programm.

Legen Sie zum Speichern der Programmdateien für dieses Buch am besten neue Ordner an und nutzen Sie die jeweilige Kapitelnummer als Ordnername. Die Datei `main.go` speichern Sie also in einem Ordner namens `Kapitel 1` in Ihrem Benutzerverzeichnis. Auf einem Mac sieht das wie folgt aus:

```
~/Kapitel 1
    |__main.go
```

Und unter Windows:

```
C:\Benutzer\IhrName\Kapitel 1
    |__main.go
```

Sobald die Datei gespeichert wurde, erscheinen Ihre eingegebenen Go-Befehle in farblicher Codierung: hellblau für bestimmte Schlüsselwörter wie `package`, `import` und `func`, orange für Zeichenketten wie `Hallo Welt!` oder `fmt` und gelb für Funktionen wie `main()` und `Println()`.

Kompilieren und Ausführen Ihres Programms

Nachdem Sie ein Programm gespeichert haben, müssen Sie es kompilieren. Erst dann kann es ausgeführt werden.

Um Ihr Programm direkt in Visual Studio Code auszuführen, klicken Sie in der Menüleiste auf Terminal | Neues Terminal. Es öffnet sich ein Kommandozeilenfenster im unteren Bereich von Visual Studio Code (siehe Abbildung 1.8).

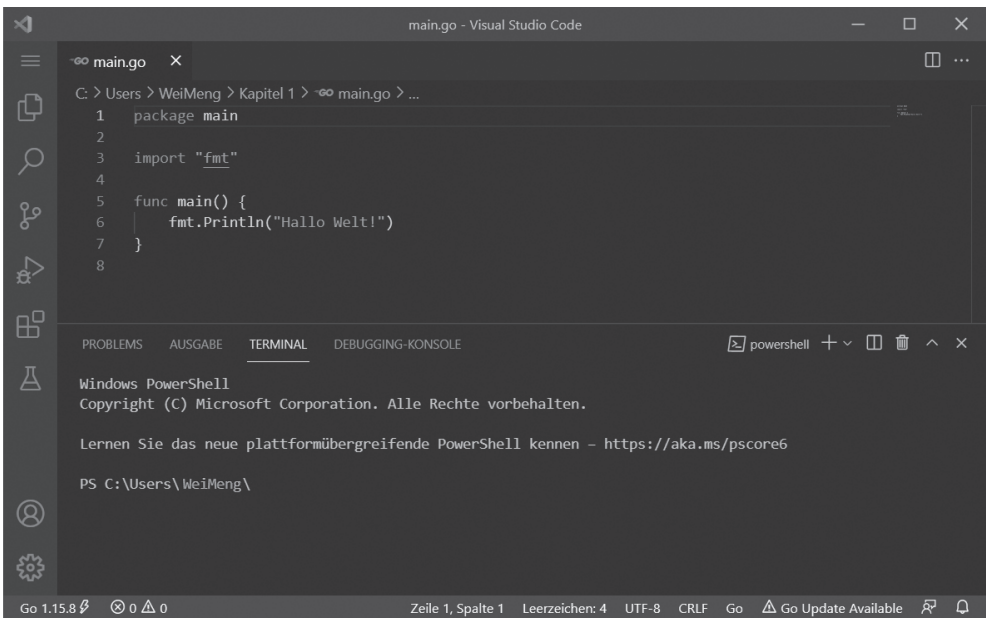

Abbildung 1.8: Die Kommandozeile kann direkt in Visual Studio Code aufgerufen werden.

Wechseln Sie nun zum Verzeichnis `Kapitel 1`. Unter macOS geben Sie dazu folgenden Befehl ein:

```
$ cd ~/"Kapitel 1"
```

Und unter Windows geben Sie diesen Befehl ein:

```
$ cd "C:\users\IhrName\Kapitel 1"
```

Jetzt können Sie die Datei `main.go` mit folgendem Befehl kompilieren:

```
$ go build main.go
```

Beim Kompilieren wird aus dem Programmcode der Datei `main.go` eine ausführbare Datei erzeugt und automatisch im Ordner `Kapitel 1` gespeichert. Um diese Datei namens `main` unter macOS auszuführen, geben Sie folgenden Befehl ein:

```
$ ./main
Hallo Welt!
```

Wird die Zeichenkette `Hallo Welt!` ausgegeben? Dann herzlichen Glückwunsch! Sie haben erfolgreich Ihr erstes Programm mit Go geschrieben.

Unter Windows erzeugt der `build`-Befehl aus Ihrem Code eine ausführbare Anwendung namens `main.exe`. Um sie auszuführen, geben Sie folgenden Befehl ein:

```
C:\users\IhrName\Kapitel 1>main.exe
Hallo Welt!
```

Da man den geschriebenen Programmcode oft direkt nach dem Kompilieren testen möchte, gibt es den Befehl `run`, mit dem Sie das Kompilieren und Ausführen in einem Rutsch erledigen:

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

Funktionsweise eines Go-Programms

Ihr erstes Go-Programm hat hoffentlich funktioniert. Aber wie eigentlich? Schauen wir uns den Code einmal Zeile für Zeile an. Die erste Zeile legt den Namen des Pakets (`package`) für dieses Programm fest:

```
package main
```

Go-Programme werden in *Paketen* organisiert. Ein Paket ist eine Sammlung aus Quelldateien, die sich alle im selben Verzeichnis befinden. In diesem Beispiel verweist `main` auf das Paket, das Sie im Verzeichnis `Kapitel 1` gespeichert haben. Wenn Sie weitere Quelldateien (`.go`-Dateien) hinzufügen, werden sie alle diesem `main`-Paket zugeordnet (mehr dazu in späteren Kapiteln).

Pakete müssen nicht zwingend den Namen `main` erhalten. Bei der Namenswahl sind Ihnen fast keine Grenzen gesetzt. Allerdings hat der Paketname `main` eine besondere Bedeutung: Alle Pakete mit diesem Namen enthalten eine Funktion namens `main()`, die als Einstiegspunkt für Ihr Programm dient. Auch der Dateiname muss nicht unbedingt mit dem Namen des enthaltenen Pakets übereinstimmen. Ich hätte meine Datei statt `main.go` auch `hund.go` nennen können – das Programm hätte trotzdem funktioniert (nur mit dem Unterschied, dass auch die ausführbare Anwendung automatisch den Namen `hund` statt `main` erhalten hätte).

Die nächste Codezeile importiert ein Paket namens `fmt`:

```
import "fmt"
```

Dieses Paket umfasst diverse Funktionen, mit denen Sie Ausgaben formatieren und an die Konsole senden können, sowie Funktionen zum Erfassen von Benutzereingaben.

Falls Ihnen einige Begriffe aus dem Programmierjargon noch nicht geläufig sind: Eine *Funktion* ist ein Block aus Codezeilen, die eine bestimmte Aufgabe erfüllen. Weitere Details zu Go-Funktionen bietet Kapitel 5.

Der nächste Codeblock stellt den Einstiegspunkt Ihres Programms dar:

```
func main() {

}
```

Da der Paketname `main` lautet, wird Ihr Programm bei der Ausführung zuerst diese `main()`-Funktion durchlaufen.

Die letzte Zeile Ihres Programmcodes ruft die `Println()`-Funktion aus dem Paket `fmt` auf, um die Zeichenkette `Hallo Welt!` in der Konsole auszugeben:

```
fmt.Println("Hallo Welt!")
```

Go-Dateistrukturen

Sie wissen nun, wie ein einfaches Go-Programm funktioniert. Aber wie werden die einzelnen Go-Dateien gruppiert und organisiert? Wie im vorigen Abschnitt erwähnt, gehören alle Dateien innerhalb desselben Verzeichnisses auch zum selben Paket. Zur Veranschaulichung fügen wir deshalb jetzt dem Verzeichnis `Kapitel 1` eine neue Datei hinzu, der wir den Namen `show_time.go` geben. (Wie Sie eine neue Datei erstellen und speichern, wurde im Abschnitt *Ihr erstes Go-Programm* erklärt.) Im Verzeichnis `Kapitel 1` befinden sich nun also folgende Dateien:

```
Kapitel 1
    |__main.go
    |__show_time.go
```

Tragen Sie folgende Codezeilen in die Datei `show_time.go` ein:

```
package main

import (
    "fmt"
    "time"
)

func displayTime() {
    fmt.Println(time.Now())
}
```

Wie Sie anhand der ersten Zeile sehen, gehört diese Datei zum Paket `main`. Wenn Sie mehr als ein Paket importieren möchten, müssen die Paketnamen von Klammern umschlossen sein. In diesem Beispiel werden die Pakete `time` und `fmt` importiert. Danach folgt eine Funktion namens `displayTime()`, die mithilfe der Funktion `Now()` des `time`-Pakets die aktuelle Uhrzeit inklusive Datum ausgibt.

Da die `displayTime()`-Funktion zum `main`-Paket gehört, kann sie auch in `main.go` aufgerufen werden:

```
package main

import "fmt"

func main() {

    fmt.Println("Hallo Welt!")

    displayTime()

}
```

Funktionen, die im selben Paket definiert sind, können aufgerufen werden, ohne dass Sie das Paket explizit importieren müssen.

Da nun zwei Dateien zum Paket `main` gehören, müssen Sie beim Kompilieren keinen Dateinamen mehr angeben. Stattdessen führen Sie den Befehl `build` innerhalb des Verzeichnisses `Kapitel 1` aus:

```
$ go build
```

Wenn Sie nun in das Verzeichnis `Kapitel 1` schauen, befindet sich dort eine neue Datei namens `Kapitel_1` (`Kapitel_1.exe` unter Windows).

Um diese Datei unter macOS auszuführen, geben Sie Folgendes im Terminal ein:

```
$ ./Kapitel_1
```

Unter Windows verwenden Sie folgenden Befehl:

```
C:\users\IhrName\Kapitel 1>Kapitel_1
```

Sie sollten nun eine Ausgabe ähnlich der folgenden sehen:

```
Hallo Welt!
2020-10-01 12:01:13.412568 +0800 +08 m=+0.000365532
```

Kompilieren von Programmen für mehrere Betriebssysteme

Bei der Installation von Go werden automatisch mehrere Go-Umgebungsvariablen eingerichtet, damit Ihre Go-Programme korrekt funktionieren. Insbesondere erkennt das Installationsprogramm bestimmte Werte der Host-Architektur und des genutzten Betriebssystems, die für die Umgebungsvariablen `GOHOSTARCH` beziehungsweise `GOHOSTOS` nötig sind. Die Werte dieser zwei Variablen geben die Zielplattform an, für die Ihre Programme kompiliert werden.

Wenn Sie sich diese Werte der Go-Umgebungsvariablen anzeigen lassen möchten, nutzen Sie den Befehl `env`:

```
$ go env
GOARCH="amd64"
GOBIN=""
GOCACHE="/Users/weimenglee/Library/Caches/go-build"
GOEXE=""
GOFLAGS=""
GOHOSTARCH="amd64"
GOHOSTOS="darwin"
GOOS="darwin"
...
...
PKG_CONFIG="pkg-config"
```

Um Ihr Programm für ein anderes Betriebssystem zu kompilieren, müssen Sie zwei weitere Umgebungsvariablen anpassen: `GOOS` und `GOARCH`. Diese beiden Variablen konfigurieren das Ziel-Betriebssystem anhand der in Tabelle 1.1 gezeigten Werte.

Betriebssystem	GOOS	GOARCH
macOS	`darwin`	`amd64`
Linux	`linux`	`amd64`
Windows	`windows`	`amd64`

Tabelle 1.1: Umgebungsvariablen für verschiedene Betriebssysteme

Um ein Programm für macOS zu kompilieren, nutzen Sie die folgenden Befehle und Optionen:

```
$ GOOS=darwin GOARCH=amd64 go build -o Kapitel_1-mac
```

Wenn Go in naher Zukunft nativ auf Apple Silicon portiert wird, wäre der Wert von `GOARCH` entsprechend `arm64`.

Um ein Programm für Windows OS zu kompilieren, sind folgende Befehle und Optionen nötig:

```
$ cd ~/"Kapitel 1"
$ GOOS=windows GOARCH=amd64 go build -o Kapitel_1-windows.exe
```

Über die Option `-o` (kurz für *Output*) können Sie einen Namen für die ausführbare Datei angeben.

Der vorgenannte Befehl kompiliert das Paket im Verzeichnis `Kapitel 1` für Windows und speichert die ausführbare Datei als `Kapitel_1-windows.exe`.

Um ein Programm für Linux zu kompilieren, nutzen Sie folgende Befehle und Optionen:

```
$ GOOS=linux GOARCH=amd64 go build -o Kapitel_1-linux
```

Falls Sie mit Go auf dem Raspberry Pi arbeiten, nutzen Sie für `GOARCH` den Wert `arm64`.

Wenn auf Ihrem Computer macOS oder Linux läuft, können Sie die für die jeweilige Plattform erstellten ausführbaren Dateien mit dem `file`-Befehl überprüfen:

```
$ file Kapitel_1-mac
Kapitel_1-mac: Mach-O 64-bit executable x86_64
```

```
$ file Kapitel_1-windows.exe
Kapitel_1-windows.exe: PE32+ executable (console) x86-64 (stripped to
external PDB), for MS Windows

$ file Kapitel_1-linux
Kapitel_1-linux: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
statically linked, Go
BuildID=bSETwZgNDR5vlulRHnzw/KNpENRt9Hipd8Nu7HGDg/v38ZPzDs35yMw33hUxoz/Y_cN
fU8fID2cCtz36hCq, not stripped
```

Vergleich von Go mit anderen Programmiersprachen

Beim Erlernen einer neuen Programmiersprache ist es oft hilfreich, einen Vergleich mit anderen Programmiersprachen anzustellen, die Sie vielleicht schon kennen. Im Idealfall helfen Ihnen Ihre vorhandenen Kenntnisse auch bei der neuen Sprache.

In diesem Abschnitt vergleiche ich Go mit zwei branchenweit besonders häufig verwendeten Programmiersprachen, nämlich mit Java und Python. Gelegentlich ist auch ein Vergleich mit C sinnvoll, da Go syntaktische Ähnlichkeiten zu dieser Sprache aufweist. Zudem wird Go oft nachgesagt, es vereine die Geschwindigkeit von C mit der Produktivität von Python.

Wenn Sie noch keine dieser Programmiersprachen in Ihrem Repertoire haben, ist das nicht schlimm. Alle hier erwähnten Funktionen werden in den weiteren Kapiteln dieses Buches ausführlich vorgestellt.

Syntax

In Sachen Syntax weist Go mehr Ähnlichkeiten mit den Programmiersprachen C und Java auf, bei denen Codeblöcke in geschweifte Klammern gesetzt werden. Python hingegen rückt die Programmzeilen ein, um verschiedene Codeblöcke auch visuell voneinander zu trennen.

Ebenso wie Python stellt Go Funktionen in den Mittelpunkt, während sich bei Java alles um Klassen dreht und sogar Funktionen in Klassen eingeschlossen werden müssen.

Im Gegensatz zu Python und Java bietet Go keine Unterstützung für die objektorientierte Programmierung (OOP) und Vererbung. Dafür gibt es in Go zur Strukturierung sogenannte Interfaces und Structs, die genau wie Klassen funktionieren.

Wie Java nutzt auch Go die statische Typisierung. Python ist eine dynamisch typisierte Sprache.

Kompilierung

Während Python und Java in Bytecode kompiliert werden, der dann übersetzt und auf einer virtuellen Maschine ausgeführt wird, liefert Go beim Kompilieren direkt Maschinencode. Aus diesem Grund ist Go besonders leistungsfähig.

Genau wie Python und Java unterstützt auch Go die automatische Speicherbereinigung (*Garbage Collection*). Bei diesem Vorgang zur Speicherverwaltung sucht der *Garbage Collector* nach Objekten, die von einem Programm nicht mehr benötigt werden, um den dadurch belegten Speicherplatz wieder freizugeben.

Python verschlingt eine ganze Menge Arbeitsspeicher. Java ist auch nicht viel besser, da Speicherressourcen dynamisch angefordert werden. Im Gegensatz dazu können Sie den Arbeitsspeicherbedarf bei Go besser steuern.

Nebenläufigkeit

Parallelisierung und Nebenläufigkeit sind in Go integriert – deswegen lassen sich Multithreading-Anwendungen sehr einfach programmieren. Sowohl Java als auch Python unterstützen die Nebenläufigkeit durch *Threading*, allerdings nicht so effizient wie Go. Tatsächlich ist die Nebenläufigkeit eines der wichtigsten Argumente, die für Go sprechen.

Bibliotheken

Alle drei Programmiersprachen bieten umfassende Unterstützung für Bibliotheken, und zwar sowohl für Standard- als auch für Drittanbieter-Bibliotheken. Besonders letztere entscheiden oft darüber, ob sich eine Programmiersprache dauerhaft etablieren kann. Aus diesem Grund ist vor allem Python seit einigen Jahren enorm beliebt – dank der Unterstützung für Drittanbieter-Bibliotheken zur Datenanalyse erhält auch die Allgemeinheit Zugang zu maschinellem Lernen und Deep Learning. In dieser Hinsicht hängt Go zwar etwas hinterher, da es die Sprache ja noch nicht so lange gibt, doch die Anzahl an verfügbaren Bibliotheken für Go nimmt stetig zu.