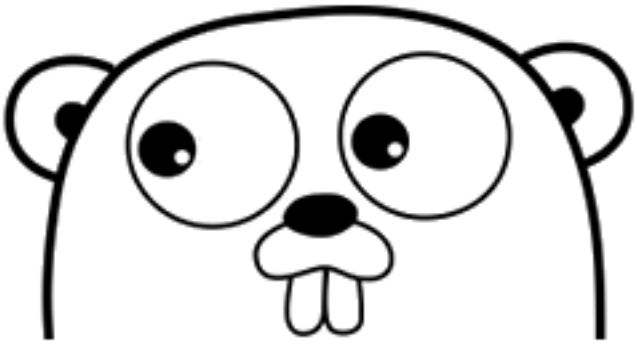

The Little Go Book

by Karl Seguin

کتاب کوچک Go

کارل سگوین

کیارش بختیار

فهرست مطالب

فصل ۱

در باره این کتاب

مجوز

آخرین نسخه

منبع آخرین نسخه در نشانی https://github.com/karlseguin/the-little-go-book در دسترس قرار دارد.

یادداشت مترجم

شاید اغراق نباشد که عنوان کنم انگیزه از ترجمه این کتاب بیشتر خودخواهانه بوده‌است. از آن‌جا که به زبان برنامه نویسی Go علاقه‌مند شدم و قصد یادگیری آن‌را داشتم، در گشت و گذار اینترنتی هنگامی که به دنبال منابع بودم با کتاب پیش رو آشنا شدم. همانگونه که در بخش مجوز مشاهده کردید، انتشار و ترجمه آن آزاد است. در نتیجه برآن شدم تا روش یادگیری حین ترجمه را تجربه کنم. از طرفی تصمیم گرفتم در نگارش کتاب نرم‌افزار قدرتمند حروف‌چینی لاتِکْ (LaTeX)را با استفاده از بسته زی‌پرشین (XePersian) به‌کار گیرم که جذابیت خاص خود را دارد. در نهایت برای کنترل منبعِ متن ترجمه، از نرم‌افزار git و همسان‌سازی آن با github در آدرس ذیل استفاده نمودم.

https://github.com/Kiarashbakhtiar/the-little-go-book-fa

کیارش بختیار، پاییز ۱۳۹۹

فصل ۲

مقدمه

همواره نوع رابطه‌ام در یادگیری زبان‌های جدید عشق و نفرت است. از یک سو، زبان‌ها برای کاری که ما می‌کنیم بسیار اساسی هستند، به طوری که حتی تغییرات کوچک می‌توانند تأثیر قابل ملاحظه‌ای داشته باشند. در لحظه‌ای خاص کلیک کردن می‌تواند تأثیر ماندگاری بر نحوه برنامه نویسی شما داشته باشد و می تواند انتظارات شما را از زبان‌های دیگر تعیین کند. از جنبه منفی، طراحی زبان به طور نسبی افزایشی است. یادگیری کلمات کلیدی جدید، نوع سامانه، روش کدگذاری و همچنین کتابخانه‌ها، جوامع و پارادایم‌های جدید کار بزرگی است که توجیه آن دشوار به نظر می‌رسد. در مقایسه با هر چیز دیگری که باید یاد بگیریم، زبان‌های جدید اغلب احساس سرمایه گذاری محدود در زمان را برای ما همره دارند.

با این اوصاف ما باید رو به جلو جرکت کنیم. خواست باید این باشد که گام‌های اساسی برداریم، زیراکه زبان‌ها پایه و اساس کار ما هستند. اگرچه این تغییرات غالباً افزایشی است، اما دامنه گسترده‌ای دارند و بر بهره‌وری، خوانایی، عملکرد، آزمون، مدیریت وابستگی، مدیریت خطا، مستندات، پروفایل، جوامع، کتابخانه‌های استاندارد و غیره تأثیر گذارند.

در اینجا یک سؤال اساسی مطرح می‌شود: **چرا Go؟** برای من دو دلیل قانع کننده وجود دارد. نخست این است که یک زبان نسبتاً ساده با یک کتابخانه استاندارد نسبتاً ساده. از بسیاری جهات، ماهیت افزایشی Go باعث ساده نمودن برخی از پیچیدگی‌هایی است که طی دو دهه گذشته شاهد اضافه شدن آن به زبان‌ها بوده ایم. دلیل دیگر این است که برای بسیاری از توسعه‌دهندگان ، زرادخانه آن‌ها را تکمیل می‌کند.

زبان Go به عنوان یک زبان سیستم ساخته شده است (به عنوان مثال، برای توسعه سیستم‌عامل‌ها و درایور دستگاه‌ها) بنابراین هدف آن، توسعه دهندگان C و C++ است. طبق گفته تیم Go ـ این مطمئناً در مورد من صادق است ـ توسعه‌دهندگان برنامه‌ها غیر از توسعه‌دهندگان سیستم، به استفاده‌کنندگان اصلی Go تبدیل شده‌اند. چرا؟ من نمی‌توانم برای توسعه‌دهندگان سیستم مقتدرانه صحبت کنم، اما برای آن دسته از ما که وب‌سایت‌ها، سرویس‌ها، برنامه‌های رومیزی و موارد مشابه ایجاد می‌کنیم، تا حدی به نیاز آشکاری وجود دارد به یک کلاس از سیستم‌هایی که در جایی بین برنامه‌های سطح پایین سیستم و برنامه های سطح بالاتر قرار دارند.

شاید این یک پیام رسان، ذخیره سازی، تجزیه و تحلیل داده های سنگین محاسباتی، رابط خط فرمان، ثبت رخدادها یا سامانه نظارتی باشد. من نمی‌دانم چه برچسبی به آن بدهم، اما در طول کار حرفه‌ای من، به دلیل اینکه سیستم‌ها همچنان از لحاظ پیچیدگی رشد می‌کنند و نیاز به همزمانی و اندازه آن نیز به همین صورت رشد می‌کند، آشکارا نیاز به سیستم‌های سفارشی از نوع زیرساخت وجود دارد. شما می توانید چنین سیستم‌هایی را با Ruby یا Python یا چیز دیگری بسازید (البته بسیاری از افراد این کار را می‌کنند)، اما این نوع سیستم‌ها می‌توانند از یک سیستم سفت و محکم با عملکرد بیشتر بهره‌مند شوند. به طور مشابه، شما می توانید از Go برای ساختن وب سایت استفاده کنید (البته بسیاری از مردم این کار را می کنند)، اما من همچنان ترجیح می‌دهم برای انجام این کار را از Node یا Rubby برای چنین سیستم‌هایی استفاده نمایم.

سطح دیگری وجود دارد که Go برتری دارد. به عنوان مثال، هنگام اجرای یک برنامه Go که کامپایل شده است، هیچ وابستگی وجود ندارد. دیگر لازم نیست که نگران باشید کاربران شما Ruby یا JVM را نصب کرده اند و اگر چنین است، چه نسخه‌ای نصب است.

به همین دلیل، Go به عنوان زبانی برای برنامه‌های رابط خط فرمان و سایر انواع برنامه های کمکی که برای توزیع نیاز دارید (به عنوان مثال، یک ثبت کننده ورود به سیستم) محبوبیت بیشتری پیدا می‌کند.

به بیان ساده، یادگیری Go استفاده کارآمد از وقت شماست. شما مجبور نیستید ساعت‌های طولانی را برای یادگیری یا حتی تسلط بر Go صرف کنید، و در نهایت به یک نتیجه عملی از تلاش خود خواهید رسید.

یادداشت نویسنده

به چند دلیل از نوشتن این کتاب دریغ کرده‌بودم. اولین مورد این است که مستندات خود Go، به ویژه Effective Go قابل اطمینان هستند.

مورد دیگر ناراحتی من از نوشتن کتابی درباره یک زبان است. وقتی کتاب The Little MongoDBرا نوشتم، از این تصور که اکثر خوانندگان، اصول پایگاه‌داده رابطه‌ای و مدل‌سازی را بدانند، مطمئن بودم. با کتاب Little Redis Book، می توانید با ذخیره‌سازی داده‌ها به صورت کلید_مقدار آشنا شوید و از آن بهره ببرید.

آنگونه که در مورد بندها و فصول پیش رو فکر می‌کنم، می‌دانم که نمی توانم همان فرضیات را بیان کنم. چه مدت زمان صرف صحبت در مورد رابط‌ها می‌کنید؟ تا بدانید برای برخی از افراد، این مفهوم جدید خواهد بود، در حالی که دیگران به بیش از آنچه رابط Goارد، نیاز نخواهند داشت. در نهایت، اگر به من بگویید که کدام قسمت‌ها خیلی کم‌عمق و یا بعضی دیگر خیلی دقیق هستند. باعث راحتی من می‌شود. این را به عنوان هزینه کتاب در نظر بگیرید.

فصل ۳

شروع

اگر می‌خواهید کمی با Go بازی کنید، باید Go Playground را بررسی کنید که به شما این امکان را می‌دهد بدون نیاز به نصب هر چیزی، کد را به صورت آنلاین اجرا کنید. این همچنین متداول‌ترین روش برای به اشتراک گذاشتن کد Go در هنگام جستجوی کمک در انجمن‌ها و مکانهایی مانند StackOverflow است.

نصب Go ساده است. می توانید آن را از مبدا نصب کنید، اما پیشنهاد می‌کنم از یکی از باینری‌های از پیش کامپایل شده استفاده کنید. وقتی به صفحه بارگیری می‌روید ، فایل نصبی مربوط به سیستم‌عامل‌های مختلف را مشاهده خواهیدکرد. اجازه دهید از این موارد اجتناب کنیم و بیاموزیم که چگونه Go را خودمان برپا کنیم. همانطور که خواهید دید، کار سختی نیست. به استثنای مثالهای ساده، Go طوری طراحی شده است که کد شما در داخل یک فضای‌کاری باشد. فضای‌کاری یک پوشه است که از زیر پوشه‌های pkg، srcو bin تشکیل شده است. ممکن است وسوسه شوید که Go را مجبور کنید از روش شما پیروی کند، هرگز این کار را نکنید. به طور معمول، من پروژه‌های خود را در داخل doc/~، قرار می‌دهم. برای مثال doc/blog/~ شامل فایل‌های بلاگ من است. برای Go فضای کاری من code/go/~ است و وبلاگ Go-powered در code/go/src/blog/~ قرار گرفته است.

به طور خلاصه، هرکجا قصد دارید پروژه‌های خود را قرار دهید، یک پوشه go با یک زیر پوشه src ایجاد کنید.

سیستم‌عامل Linux / OSX

فایل tar.gz را بر اساس پلتفرم خودتان بارگیری کنید. در OSX، به احتمال زیاد علاقه‌مند هستید از go#.#.#.darwin--amd64 osx10.8.tar.gz استفاده نمایید که در آن #.#.# آخرین نسخه Go است.

با استفاده از دستور ذیل

```
tar -C /usr/local -xzf go#.#.#.darwin-amd64-osx10.8.tar.gz
```

فایل را در مسیر /usr/local از حالت فشرده خارج کنید.

دو متغیر محیطی ذیل را تنظیم کنید:

۱. عبارت GOPATH به محیط کاری شما اشاره می‌کند. مثلاً برای من مسیر $HOME/code/go است.

۲. باید باینری Go را به PATH خود اضافه کنیم.

با استفاده از دستورات ذیل در shell می‌توان این کار را انجام داد:

```
echo 'export GOPATH=$HOME/code/go' >> $HOME/.profile
echo 'export PATH=$PATH:/usr/local/go/bin' >> $HOME/.profile
```

به منظور اعمال تغییرات می‌توانید پوسته خود را بسته و دوباره باز کنید، یا می‌توانید دستور source␣$HOME/.profile را اجرا کنید. دستور go␣version را تایپ کنید تا خروجی مانند go␣version␣go1.3.3␣darwin/amd64 نمایش داده شود.

سیستم‌عامل Windows

آخرین فایل zip را بارگیری کنید. اگر در یک سیستم ۶۴ بیتی هستید، به احتمال زیاد علاقه‌مند هستید از go#.#.#.darwin-amd64-windows-amd64.zip استفاده نمایید که در آن #.#.# آخرین نسخه Go است.

آن را در محلی که انتخاب کرده اید از حالت فشرده خارج کنید. پوشه c:\Go انتخاب مناسبی خواهد بود.

دو متغیر محیطی ذیل را تنظیم کنید:

۱. عبارت GOPATH به محیط کاری شما اشاره می‌کند. برای مثال می‌تواند مسیر c:\users\goku\work\go باشد.

۲. مسیر c:\Go\bin را به PATH سیستم‌عامل خود اضافه کنید.

متغیرهای محیطی را می‌توان از طریق گزینه Environment␣Variables در برگه Advanced صفحه System در Control Panel تنظیم کرد. برخی از نسخه‌های ویندوز این کنترل را از طریق گزینه Advanced␣System␣Settings موجود در System که در Control Panel قرار دارد، ارائه می‌دهند. یک command prompt باز کنید و عبارت go␣version را تایپ کنید. امیدوارم خروجی که دریافت می‌کنید مانند go␣version␣go1.3.3␣windows/amd64 باشد.

فصل ۴

مبانی

Go یک زبان کامپایلری، نوع داده استاتیک و با یک نحو شبیه C و دارای زباله‌روب است. معنی آن چیست؟

کامپایل

کامپایل فرآیند ترجمه به زبان سطح پایین‌تر یا اسمبلی، برای کدی است که می‌نویسید (همانطور که در مورد Go وجود دارد)، که در برخی از زبان های میانی دیگر (مانند جاوا و C#) وجود دارد.

کار با زبانهای کامپایل شده می‌تواند ناخوشایند باشد زیرا کامپایل کند است. اگر مجبور باشید چند دقیقه یا چند ساعت در انتظار کامپایل کد باشید، تکرار سریع آن دشوار است. سرعت کامپایل یکی از اهداف اصلی طراحی Go است. این برای افرادی که در پروژه های بزرگ کار می‌کنند خبر خوبی است و همچنین کسانی که به چرخه بازخورد سریع ارائه شده توسط زبان‌های تفسیر شده عادت دارند.

زبانهای کامپایل شده معمولاً سریعتر اجرا می شوند و قابلیت اجرایی را می‌توان بدون وابستگی‌های اضافی اجرا کرد. این برای زبانهایی مانند C، C++ و Go که مستقیماً به اسمبلی کامپایل می شوند صدق می‌کند.

ایستایی نوع داده

نوع ثابت بودن به معنای این است که متغیرها باید از نوع مشخص باشند (bool، string، int، byte[] و غیره). این امر یا با تعیین نوع، هنگامی که متغیر اعلام می‌شود حاصل می شود یا در بسیاری از موارد، اجازه داده‌می‌شود که کامپایلر نوع را استنباط کند (به زودی به مثال‌ها خواهیم پرداخت).

در مورد نوع داده ایستا چیزهای زیادی می توان گفت، اما من معتقدم که این با دیدن کد بهتر درک می‌شود. اگر شما به نوع داده پویا در زبان‌ها عادت دارید، ممکن است با مشکل روبرو شوید. شما اشتباه نمی‌کنید، اما مزایایی وجود دارد، به ویژه هنگامی که نوع داده ایستا را با کامپایل باهم در نظر داشته باشید. این دو بیشتر مواقع باهم تلفیق می‌شوند. درست است که وقتی یکی را داشته باشید، معمولاً دیگری را نیز دارید اما این یک قانون همیشگی نیست. با استفاده از یک سیستم نوع داده سفت و محکم وجود داشته‌باشد، کامپایلر قادر است مشکلات فراتر از اشتباهات صرفی را تشخیص دهد و همچنین بهینه‌سازی‌های بیشتری نیز انجام دهد.

نحو C مانند

گفتن اینکه یک زبان دارای نحوی C مانند است، به این معنی است که اگر شما به هر زبان C مانند، نظیر JavaScript، Java، C ++، C و C# آشنا باشید، با Go غریبه نخواهید بود- حداقل به صورت سطحی. به عنوان مثال، && به عنوان AND boolean استفاده می‌شود، == برای مقایسه برابری، } جهت شروع و { برای پایان یک دامنه به‌کار گرفته می‌شود و شاخص آرایه از ۰ شروع می‌شود.

نحو C مانند نیز به معنی خط‌های پایان یافته با کاراکتر ; و پرانتزهای اطراف شرط‌ها است. Go از هر دو مورد بی‌نیاز است، اگرچه هنوز از پرانتز برای کنترل تقدم عملیات استفاده می‌شود. به عنوان مثال ، دستور if به این شکل است:

```
 if name == "Leto" {
  print("the spice must flow")
 }
```

و در موارد پیچیده‌تر، پرانتزها هنوز هم مفید هستند:

```
if (name == "Goku" && power > 9000) || (name == "gohan" && power < 4000)
{
 print("super Saiyan")
}
```

فراتر از این، Go بسیار نزدیکتر به C است تا C# یا Java- نه تنها از نظر نحو، بلکه از نظر هدف نیز این‌چنین است. این در سختی و سادگی زبان منعکس شده است که امیدوارم همراه با یادگیری آن کاملاً مشهود شود.

زباله‌روب [۱]

برخی از متغیرها، هنگام ایجاد، یک روش آسان برای تعریف دارند. به عنوان مثال، یک متغیر محلی برای یک تابع، وقتی از بین می‌رود که از تابع خارج شویم. در موارد دیگر، چندان واضح نیست ـ حداقل برای یک کامپایلر. به عنوان مثال، تعیین طول عمر یک متغیر که توسط یک تابع برگردانده می‌شود یا توسط متغیرها و اشیا دیگر به آن ارجاع می شود، مشکل است. بدون جمع آوری زباله، توسعه‌دهندگان وظیفه دارند حافظه مرتبط با چنین متغیرهایی را در نقطه‌ای که توسعه‌دهنده می‌داند دیگر به متغیر مورد نظر نیاز نیست، آزاد کنند. چگونه؟ در C شما باید از ; (`free(str)`) برای آزادسازی حافظه استفاده کنید.

زبان‌هایی که دارای زباله‌روب هستند (به عنوان مثال، روبی، پایتون، جاوا، جاوا اسکریپت، سی شارپ، Go) قادر به پیگیری این موارد هستند و وقتی دیگر از متغیری استفاده نمی‌شود آن را آزاد می‌کنند. استفاده از زباله‌روب سربار اضافه می‌کند، اما در عوض تعدادی از اشکالات ویرانگر را از بین می‌برد.

---

[۱] Collected Garbage

اجرای کد Go

بیایید سفر خود را با ایجاد یک برنامه ساده و یادگیری نحوه کامپایل و اجرای آن آغاز کنیم. ویرایشگر متن مورد علاقه خود را باز کرده و کد زیر را بنویسید:

```
package main
func main() {
 println("it's over 9000!")
}
```

فایل را به عنوان main.go ذخیره کنید. در حال حاضر، می توانید آن را در هر کجا که می‌خواهید ذخیره کنید. برای نمونه‌های بی‌اهمیت نیازی به کار در فضای کاری Go نداریم.

سپس ،یک پوسته یا خط فرمان باز کرده و مسیر را به محلی که فایل را ذخیره کرده‌اید تغییر دهید. برای من، این به معنای تایپ cd␣~/code است.

در آخر ،برنامه را با وارد دستور ذیل اجرا کنید:

```
go run main.go
```

اگر همه چیز کار کرد، شما باید عبارت !it's over 9000 را ببینید.

اما صبر کنید ، در مورد مرحله کامپایل چه می‌کنید؟ go run یک دستور مفید است که کد شما را کامپایل و اجرا می‌کند. برای ساخت برنامه از یک فهرست موقتی استفاده می‌کند، آن‌را اجرا و سپس خود را تمیز می‌کند. با اجرای دستور زیر می‌توانید مکان فایل موقت را مشاهده کنید:

```
go run --work main.go
```

برای صرفاً کامپایل کردن کد، از دستور go build استفاده کنید:

```
go build main.go
```

این یک main قابل اجرا ایجاد می‌کند که می‌توانید آن‌را اجرا کنید. در Linux / OSX فراموش نکنید که باید فایل اجرایی را با پیشوند ./ تایپ کنید، بنابراین برای اجرا باید دستور ./main را تایپ کنید.

هنگام توسعه، می توانید از go run یا go build استفاده کنید. با این وجود، هنگامی که کد خود را پیاده‌سازی می‌کنید، می‌توانید یک باینری از طریق go build ایجاد کرده و آن را اجرا کنید.

Main

امیدوارم، کدی که ما تازه اجرا کردیم قابل درک باشد. ما یک تابع ایجاد کردیم و یک رشته را با تابع println داخلی چاپ کردیم. آیا go run می‌دانست چه چیزی را اجرا کند با اینکه فقط یک انتخاب وجود داشت؟ خیر. در Go، نقطه ورود به یک برنامه باید تابعی باشد به نام main در یک بسته main.

در فصل بعدی درباره بسته‌ها بیشتر صحبت خواهیم کرد. در حال حاضر، در حالی که ما به درک اصول Go توجه داریم، همیشه کد خود را در بسته main می‌نویسیم.

در صورت تمایل می‌توانید کد را تغییر و نام بسته را تغییر دهید. کد را از طریق go run اجرا کنید و باید یک خطا دریافت کنید. سپس، نام را به main تغییر دهید اما از نام تابع دیگری استفاده کنید. در این حالت باید یک پیام خطایی متفاوت ببینید. سعی کنید همان تغییرات را اعمال کنید اما به جای آن از go build استفاده کنید. توجه داشته باشید که کد کامپایل می‌شود، فقط نقطه ورودی برای اجرای آن وجود ندارد.

فراخوانی بسته‌ها [۱]

Go دارای تعدادی توابع داخلی مانند println است که بدون مرجعی قابل استفاده هستند. بدون استفاده از کتابخانه استاندارد Go و در نهایت استفاده از کتابخانه‌های شخص ثالث[۲]، نمی توانیم خیلی کارها را انجام دهیم. کلمه کلیدی import برای شناساندن بسته‌های استفاده شده توسط کد موجود در فایل برنامه‌تان استفاده می‌شود.

بیایید برنامه خود را به شکل زیر تغییر دهیم:

```
package main

import (
"fmt"
"os"
)
func main() {
 if len(os.Args) != 2 {
  os.Exit(1)
 }
 fmt.Println("It's over", os.Args[1])
}
```

که می توانید از طریق دستور زیر آن‌را اجرا کنید:

```
go run main.go 9000
```

همانگونه که مشاهده می‌کنید ما از دو بسته استاندارد Go استفاده می کنیم یعنی: fmt و os. ما همچنین یکی دیگر از توابع داخلی یعنی len را معرفی کرده‌ایم. len اندازه یک رشته یا تعداد مقادیر دیکشنری یا همانطور که در اینجا می‌بینیم تعداد عناصر آرایه را برمی‌گرداند. اگر از خود می‌پرسید که چرا ما انتظار داریم ۲ آرگومان داشته باشیم، دلیل آن این است که اولین آرگومان ـ در شاخص ۰ ـ همیشه مسیر اجرایی در حال اجرا است. (برنامه را تغییر دهید تا چاپ شود و خودتان ببینید.).

احتمالاً متوجه شده‌اید که پیشوند نام تابع را با نام بسته به‌کار می‌بریم، به عنوان مثال، fmt.Println. این با بسیاری از زبان‌های دیگر متفاوت است. در فصل‌های بعدی درباره بسته‌ها بیشتر خواهیم آموخت. در حال حاضر، دانستن نحوه وارد کردن و استفاده از بسته‌ها، شروع خوبی است.

Go در مورد فراخوانی بسته‌ها سخت‌گیری می‌کند. اگر بسته‌ای را وارد کنید اما از آن استفاده نکنید، کامپایل نمی‌شود. سعی کنید موارد زیر را اجرا کنید.

```
package main

import (
"fmt"
"os"
)
func main() {
}
```

---

[۱] Impots

[۲] third-party

برای fmt و os که فراخوانی شده‌اند ولی استفاده نشده‌اند باید دو خطا دریافت کنید. آیا این می‌تواند آزار دهنده باشد؟ در پاسخ باید گفت، کاملاً آزار دهنده است. با گذشت زمان، شما به آن عادت خواهید کرد (هر چند هنوز آزار دهنده خواهد بود). رفتار Go در این مورد سختگیرانه است زیرا فراخوانیِ بلااستفاده می‌تواند کامپایل را کند کند. مسلماً مشکلی که اکثر ما تا این حد با آن روبرو نیستیم.

نکته دیگری که باید به آن توجه کنید این است که کتابخانه استاندارد Go به خوبی و به طور کامل مستند شده‌است. به طور مثال برای کسب اطلاعات بیشتر در مورد تابع Println که ما استفاده کردیم می‌توانید به https://golang.org/pkg/fmt/#Println مراجعه کنید. می‌توانید بر روی عنوان آن قسمت کلیک کرده و کد منبع را مشاهده کنید. همچنین، برای کسب اطلاعات بیشتر در مورد قابلیت‌های قالب‌بندی Go، به بالا بروید.

اگر در جایی بدون دسترسی به اینترنت گیر کرده‌اید، می توانید مستندات را به صورت محلی از طریق زیر دریافت کنید:

```
godoc -http=:6060
```

و در مرورگر خود آدرس http://localhost:6060 را وارد کنید.

متغیرها و اعلان‌ها

توابع و اعلان‌ها

قبل از ادامه

فصل ۵

ساختارها

فصل ۶

نقشه‌ها، آرایه‌ها و برش‌ها

فصل ۷

ساختار کد و رابط‌ها

فصل ۸

ریزه‌کاری‌ها

فصل ۹

همزمانی

فصل ۱۰

نتیجه‌گیری