go version ===> نمایش ورژن نصب شده بر روی سیستم

which go ===> مسیر نصب بر روی سیستم را نمایش میدهد

go mod init GoLang # your\_module\_mane

go run name\_file.go ====> اجرای فایل مورد نظر

go help ===> نمایش راهنما

go help build ====> نمایش راهنمای دستور build

go buld name\_file.go ====> ساخت فایل اجرایی

go env

$GOPATH

cd $GOPATH

ساختار golang به این شکل است که اول برنامه را میسازد و بعد آن را اجرا میکند پس در نظر داشته باشید که اگر خطایی در برنامه شما وجود داشته باشد اصلاً برنامه شما اجرا نمیشود

ماننده زبان جاوا فانکشن اصلی main میباشد و هر چیزی که قرار است اجرا شود باید در main قرار بگیرد

در این زبان باید مقداری که ساخته می‌شود حتماً از آن استفاده شود وگرنه ارور دریافت خواهید کرد

constants که با کلمه کلیدی const آن را تعریف میکنیم متغیری هست که نمیتوان آن را تغییر داد

اگر میخواهید متغیری public باشد و در جاهای دیگر بتوانید از آن استفاده کنید ( یا در اصطلاح global باشد ) باید حرف اول آن بزرگ باشد و اگر حرف اول آن کوچک باشد فقط به صورت localی میتوان از آن استفاده کرد

هنگام تعریف متغیر از : استفاده می‌شود . : به معنی تعریف type متغیر میباشد

به طور کلی garbage collector در زبان go وجود دارد به این معنی که وقتی کار با مموری تمام شد garbage collector موارد اضافی را از مموری پاک میکند مموری یکی از resource هایی است که میتوان از آن استفاده کرد و garbage collector فقط برای مموری میباشد برای بقیه resource ها ماننده فایل‌ها و سوکت ها و … باید از defer استفاده شود این defer اگر error یا پنیک هم داشته باشیم بازهم اجرا خواهد شد

برای درک بهتر pointer لازم است که بدانیم یک متغیر از چه قسمت‌هایی تشکیل شده است:  
 اولین قسمت نام آن متغیر میباشد   
 دوم قسمت آدرس میباشد که آدرسی از خانه حافظه را در خود ذخیره میکند   
 سومین قسمت مقداری از که در آن آدرس خانه حافظه ذخیره شده است   
 چهارمین قسمت type متغیر میباشد

اگر متغیری بخواهد به آدرس متغیر دیگری اشاره کند باید از & استفاده کنیم و وقتی بخواهیم مقداری در آن آدرس بریزیم لازم است که در pointer آن بریزیم که از \* استفاده میکنیم ( با این حساب دو متغیری ساخته می‌شود که هر دو به یک خانه از حافظه اشاره دارند )

وقتی یک تابع پوینتر میگیره چنین اتفاقی میوفته. ما آدرس رو داریم میفرستیم و باید تغییراتی که میخوایم رو روی اون آدرس تغییر بدیم. یعنی روی value ورودی تابعمون نه روی آدرس متغیری که به عنوان ورودی تعریف کردیم.

اگر ما یک متغیری را به یک تابعی پاس بدیم، زبان میاد یک کپی از اونا میگیره و در داخل تایع ازش استفاده

در زبان گو panic همانند exception به معنای خروج از برنامه در شرایط غیر عادی می باشد. panic در ۲ حالت زیر پیش می‌آید

خطاهای در زمان اجرای برنامه

فراخوانی تابع panic توسط برنامه نویس در بخش های مختلف برنامه

slice ها درواقع همان آرایه ها هستند و map ها در‌واقع همان آبجکت هستند ( به صورت key و value )

map در‌واقع خود pointer است و خواصی ماننده pointer دارد

یک struct (ساختار) به مجموعه‌ای از فیلدهای مرتب با هم که یک واحد منسجم (unit) را تشکیل می‌دهند اشاره دارد. به هر عضو از این unit یک فیلد field گفته می‌شود. ساختار یک نوع تعریف شده توسط کاربر (user-defined) است که در مواردی که منطقی باشد که داده ها را به جای مقادیر جداگانه در یک واحد قرار دهیم استفاده می شود.

در struct میتوان توابع را با استفاده از composition به هم منتقل کنیم و نمیتوانیم این کار را با متغیر ها انجام دهیم

اینترفیس‌ها انتزاع‌هایی هستند که رفتار یک نوع خاص را تعریف می‌کنند، اما جزییات شیوه پیاده‌سازی آن رفتار را توصیف نمی‌کنند.

Type assertion یک حالت خاصی از conversion میباشد در‌واقع هنگامیکه قرار است type متغیری به interface تغییر کند type assertion رخ میدهد ( هنگامیکه type یک پارامتر را از نوع interface قرار میدهیم به این معنی میباشد که همه چیز قبول است )

در interface به متغیرها دسترسی نداریم (‌ interface برای این ساخته شده است که بگوییم در آن این متدها وجود دارد )

type های ما دو مدل struct و interface میباشد که در struct فقط متغیر ها و در interface فقط توابع وجود دارد





