Goda packaging yapısı genelde modüller üzerinden ilerler. Hem paket var hemde onunla ilintili modul yapısı var.

Go'yu nasıl daha iyi yazarım diyorsak https://go.dev/doc/effective_go buradakileri yapabiliriz gocular bunu nasıl kullanırlar. Best practice olarak.

Godaki standat paketler https://pkg.go.dev/std

Go baya command orianted bir dil. Commant line'ı sevmemiz lazım.

Golang static bir dildir

**Statik olarak yazılan diller**

Derleme zamanında bir değişkenin türü biliniyorsa, bir dil statik olarak yazılır. Bazı diller için bu, programcı olarak her değişkenin ne tür olduğunu belirtmeniz gerektiği anlamına gelir (örn: Java, C, C ++); diğer diller bir tür çıkarsama şekli sunarlar , tür sisteminin bir değişkenin türünü çıkarma kabiliyeti (örneğin: OCaml, Haskell, Scala, Kotlin)

Buradaki ana avantaj, her türlü kontrolün derleyici tarafından yapılabilmesidir ve bu nedenle çok erken aşamada birçok önemsiz hata yakalanmaktadır.

Örnekler: C, C ++, Java, Rust, Go, Scala

**Dinamik olarak yazılan diller**

Tür, çalışma zamanı değerleriyle ilişkilendirilmişse ve değişkenler / alanlar / vb. Olarak adlandırılmamışsa, bir dil dinamik olarak yazılır. Bu, bir programcı olarak biraz daha hızlı yazabileceğiniz anlamına gelir çünkü her seferinde tür belirtmeniz gerekmez ( tür çıkarımlı statik olarak yazılmış bir dil kullanmadıkça ).

Örnekler: Perl, Ruby, Python, PHP, JavaScript

Çoğu komut dosyası dili bu özelliğe sahiptir, çünkü yine de statik tür denetimi yapacak bir derleyici yoktur, ancak bir değişkenin türünü yanlış yorumlayan yorumlayıcının neden olduğu bir hata ararken kendinizi bulabilirsiniz. Neyse ki, komut dosyaları küçük olma eğilimindedir, bu nedenle hataların gizlenecek çok yeri yoktur.

Dinamik olarak yazılan dillerin çoğu, tür bilgisi sağlamanıza izin verir, ancak bunu gerektirmez. Halen geliştirilmekte olan bir dil olan Rascal , işlevler içinde dinamik yazmaya izin veren ancak işlev imzası için statik yazmayı zorunlu kılan karma bir yaklaşım benimser.

Go complier bir dildir yani  Programlama dili ile yazıyorsunuz ve o program makina diline çevriliyor. Ama bu mütercim tercüman usulü işlem yürütülürken yapılıyor. Yani; elinizdeki kodu çalıştırmak istediğiniz anda bir yandan makina diline çevriliyor bir yandan o kod çalıştırılıyor. Normalde derlemeli dillerde derleme işlemini yapan programa derleyici (compiler) demiştik, yorumlama işini yapana da yorumlayıcı (interpreter) deniyor onlar bir complier işleminden geçmez. Değişkenlerin tipi varsa complier time'da integer tanımladım ve bu runtimeda değişmiyosa buna static type dil deriz. Static type diller insanı çok daha sık boğaz eden typlerla ilgili çok kontrol yapan olduğu diller olduğu için hataya daha az yer verirler. Complier timeda yaptıları dönüşümlerle runtimeleri aşırı derecede robos

ve hızlı olma eğilimindedir. Sistem dilleri genelde böyle olur go'da böyle bir dil c ,c++ gibi. İnterpeter diller daha çok typles olması eğiliminde nasıl yani iki integer double için akın, naber string bunu bilmiyomu runtime bunu evet biliyo ama burada typles olan şey data değil. Typles dillerde değişkenlerin type tipik olarak olmaz bu ne demektir ben bir i ye int atarım 2 dk sonra bool atarım 5 dk sonra byte atarım runtime bunu yönetir. O anlamda type baskısı olmadığı için öğrenmesi kullanması daha kolay dillerdir. Eleştirilen tarafı runtime performansları ve hataya yatkın olmalarıdır. java hem complier hemde İnterpeter dildir.

Type inferencedir yani greeting:="Ali" yapsanda o onun tipini bilir.

Go'da fiellar var fiellar paketlere bağlı paketlere modüllere bağlı.

Go da modüllere ulaşıyoruz oradanda paketlere.

Var keyword tipik olarak tanımlamak için en genel halde bu bir constant değilse bir identifier tanımlıyoruz.

İhtiyacımıza göre bir initilazation assegment operatorunude solda yazabiliriz.

Var name string= "kerim" initilazier edilebilirler.

---

İstersek main func dışarıdan parametre alabilir. Os.Args[1] ile dışaradan ayşe yazarsak bize funca gider ve selam ayşe der..  greeting /argument/greet.go

Genelde dillerde scannerlar vardır bunlarla string double vb gibi yapılar alır okursunuz temelde bir satırda girdiğim değişkeni bit satıra atar ilk slayt 27. Sayfa.

```go
package main

import (
    "fmt"
    "os"
)

func main() {
    name := os.Args[1]
    greeting := createGreet(name)
    fmt.Printf( format: "%s\n", greeting)
}

func greet(name string) {
    fmt.Printf( format: "Selam %s :)\n", name)
}

func createGreet(name string) string {
    return "Selam " + name + " :)"
}
```

**Araştırmalar**

**Side effect nedir?**

*Side Effect; uygulamalarda yapılan sorgulamalar neticesinde verilerde herhangi bir değişikliğin oluşup oluşmaması durumudur. Örneğin, Select * FROM Personeller sorgusunu ele alırsak eğer ilgili sorgu her ne kadar çalıştırılırsa çalıştırılsın herhangi bir kaynakta değişikliliğe sebep olmayacağından dolayı yan etkisi olmayan(no side effect) bir sorgudur.*

*Burada şöyle bir metafor ile düşüncemizi destekleyebilir ve kavramın anlaşılmasını kolaylaştırabiliriz; bir ilacı kullandığınızda genellikle istenmeyen kötü etkenlerini 'yan etkiler' olarak nitelendirmekteyiz. Halbuki ilacı kullanmanın temel amacı beklenilen/talep edilen/ideal olan etkiyi elde etmek ve şifaya kavuşmaktır. Bu mantık ile fonksiyonların açısından baktığımızda, hedeflediği operasyonu gerçekleştirmesi için çağrılan fonksiyon ideal olan/beklenen sonucu üretebilmek için başka kaynakları yahut verileri değiştiriyorsa eğer işte bu durum yazılımsal açıdan bir 'yan etki' olarak nitelendirilmektedir.*

*Yani kısaca anlayacağınız side effect bir işlem esnasında başka bir durumun, değişkenin, nesnesinin yahut değerin değiştirilmesine atıfta bulunan kavramdır.*

*ÖZET:* ***Programlamada; bir fonksiyonun kapsamı dışındaki bir değişkeni, değeri veya objeyi değiştirmesi yan etkidir.***

**Kaynak ve daha fazlası** https://www.gencayyildiz.com/blog/yazilimda-side-effectyan-etki-kavrami-nedir/#:~:text=Side%20Effect%3B%20uygulamalarda%20yap%C4%B1lan%20sorgulamalar,bir%20de%C4%9Fi%C5%9Fikli%C4%9Fin%20olu%C5%9Fup%20olu%C5%9Fmamas%C4%B1%20durumudur.

**Pure Functions?**

Aynı girdiler için her zaman aynı sonucu verecek.

Herhangi bir side effect (yan etkisi) olmayan

Metotun döndüğü değer sadece metodun parametreleri ile alakalıysa

Metotun, metot dışındaki değişkenler ve metotlardan etkilenmemesi gerekir.

Metota gönderilen parameterin değerlerinin metot içerisinde değişmemesi gerekir.

https://metinalniacik.medium.com/pure-function-nedir-40477bc57d27

https://www.cihanyakar.com/pure-function-ve-idempotent-function-kavrami/

# Mutable ve Immutable nedir ?

**Immutable (değişmez)**, nesneler bir kez oluşturulduktan sonra içeriği değiştirilemeyen sınıflardır. Tam tersi olarak, değiştirilebilen sınıflar da **Mutable (değişebilir)** sınıflardır. Kısacası Immutable nesneler değişmeyen nesnelerdir. Onları oluşturursun, fakat onları değiştiremezsin. Bunun yerine, değişmez bir nesneyi değiştirmek isterseniz, onu klonlamanız ve oluştururken klonu değiştirmeniz gerekir.
Immutable nesneler, çok iş parçacıklı(multi-threaded) ortamlarda ve streamlerde kullanışlıdır. Değişmeyen nesnelere güvenmek harikadır. Başka bir thread'in nesnesini değiştiren bir iş parçacığının neden olduğu hatalar olabilir. Immutable nesneler, bu sorunların tümünü çözmüş olacaktır.
Brian Goetz " **Java Concurrency in Practice** " kitabında bunu daha ayrıntılı olarak açıklamıştır.
Java'da yaygın olanlarından örnek verecek olursak **String** ve tüm ilkel sarmalayıcılar **(Integer, Long, Double, Byte….), BigDecimal, BigInteger** immutable sınıflardır. **Date**, **StringBuilder** mutable sınıflardandır.
Kaynak: https://koraypeker.com/2018/11/29/mutable-ve-immutable-nedir/

---

greeting:= hem declaration hem assigment için kullanılıyor

Go'da excepitona karşılık gelen farklı yapılar var Errorlar ve Panicler. Errorları biz yönetiriz Panicler ise yapmamamız gereken şeyler.

**GO OBJECT ORIANTED'MIDIR?**

Nesneler bilirler ve yaparlar. Dil bazında class ve benzeri yapılarda halledilir her dilde olmayabilir. Genel olarak bizim abstraction'u düşündüğümüz şeyi encapsule etmesini sağlayan yapıya. Biz dillerde class gibisinden yapılarla bunu implemente ediyoruz.

İnheritance objeler arasındaki benzerlikleri ifade etmeye yarayan yapı.

polimorfizm benzer objelerin benzer şekilde davranmalarını sağlayan özellik.

Go'nun içinde temel abstraction class değil structlardır.

Class yok bir abstraction var data abstraction. Encapsulation veri ile veri üzerinde çalışan davranışları bir araya getirip dışarıdan yalıtmak. Bizim bildiğimiz anlamda bir encapsulation yok. Varsa bile daha bölünmüş olarak var. Fonksiyonlarda'da davranışları encapsule edip struclarla ilişkilendiriyoruz. Bir yerde hem datayı hem davranışı koymayız bunları ayrı ayrı iki farklı encapsulation yapısıyla hallederiz. Sonrada birbiriyle ilişkilendirebiliriz. Strucların objeleri instanceleri oluşabilir. Onlarla ilişkilendirdiğimiz fonlsiyonları'ki artık onlara metot diyoruz bir functionu structla ilişkilendirdiğimizde o metot oluyo structun instance üzerinde çağırabiliriz. Temelde OOP go'da implemete edilme şekli böyle. Bir fonksiyon tanımlıyorum onu structla ilişkilendirdiğim zaman bir metot oluyor.

```go
// greeting/oo/greet.go
package main

import (
  "fmt"
)

type Person struct {
  name string
}

func (p Person) greet() string {
  return "Selam " + g.name + " :)"
}

func main() {
  greeter := Person{"Nihal"}
  var greeting = greeter.greet()
  fmt.Printf("%s\n", greeting)
}
```

Bunu fonksiyon tanımlayıp metot yapmasaydım. P Person'u kaldıracaktım o halde fonksiyon olarak kullanabilecektim. Structları extend edip inheritance yapmak söz konusu değil. İnterfacelerde var interfaceleri devralmak gibi.

**FUNCTİONAL PROGRAMİNG NEDİR?**

Functional programing dediğimizde aklımızda ilk gelen şey Immutable olmalı değişmeme prensibi olmalı. Functional programingin en temel yöntemi olabildiğince az değişiklik yapmak. Yada hiç durum değişiklikleri yapmamak. No side efectin olması fonksiyonların birbirine parametre olarak geçilebilmesi aslında bizim Immutable'nin sağlama mekanizmaları. İmmuyu sağlamadıktan sonra bu fonksiyonların birbirine geçirilmesininde bir anlamı yoktur. Temelde immutuabilty. Olabildiğince az değişiklik yaparak kodlama yapma yöntemi. Func programing paradigması.

**Fonksiyonel Programlamanın Şartı Kaçtır?**

Fonksiyon parametreleri hariç veri üzerinde işlem yapmaz.

Fonksiyon parametrelerini veya dışındaki veriyi değiştirmez.

Fonksiyon sadece değer döndürerek sonuç üretir.

Fonksiyon sadece bir mantıksal işlevi yerine getirir.

Fonksiyon aynı girdilerle hep aynı çıktıyı üretir.

Kaynak: https://turerkan.medium.com/nedir-bu-fonksiyonel-programlama-73c43254d8f4

Fonksiyon mümkün tüm girdiler için mutlaka mantıklı çıktılar üretir.

```
y = f(x)


int F(int x, int y){
    int z = x * Y;
    return z;
}
```

bu bir functional programinge örnek. F'in herhangi bir yan etkisi yok. Bu fonksiyonun bütün yaptığı değişiklikler ve işler sadece kendisine geçilen x ve y paremetreliyle alakalı. İçinde bir sürü iş yapsan bile metodun sadece x,y ve onlardan oluşturduğumuz z'yi değiştiriyorsak hala pure functionaldır çünkü hala tüm değişiklikler fonksiyonun içinde kalır ve dışarıda bir şey olmaz.

Bir fonksiyon kendi scopu içindeki şeyleri değiştirebilir.

```
int m = 5;
int F(int x, int y){
    int z = x * Y * m++;
    z *= 2;
    //bir suru is yapıyorum.
    return z;
}
```

Burada dışarıdan m=5 alıyo buraya kadar sıkıntı yok ama içeride m++ olup m'nin değeri değiştiği için bu functional programinge girmez pure functional olmaz. Const gibi bir şey olsaydı mesela m değiştirilmeseydi veya içeride ++ koymasaydık bu pure functional olurdu.

İnt m=5 olan global varilable sıkıntıdır. m diyelimki bir arabanın vitesi bir fonksiyon m'yi değiştiriyor arttırıyo azaltyıo. Vites değiştiği zaman motorun diğer kısmıyla alakalı başka işlemlerde olmalı. O başka yerler bu vites değişikliğini anlamak zorundalar. O halde ne yapılabilir. Java'nın yaptığı gibi encaspsulation yapılabilir. Javada veya C# da global varilable yoktur. Neden yok bundan dolayı. M yi bir classın içine koyacağız private yapacağız gerekirse setter getter koyacağız m'yi değiştirmek isteyen vatandaş m'yi direk değiştiremeyecek bir class'ın fonksiyonlarını kullanarak değişecek. Bu kontrollü bir değişimdir. Değişimden kaçamazsın. O zaman kontrollü şekilde değiştiririz.

Go'da global varilableler vardır ama bunun sorumluluğu developdadır. m'yi gidip biri değiştiriyorsa mekanizmanın diğer taraflarıda buna uygun davranmalı.

First Code

```go
// greeting/functional/greet.go

...
func main() {
  greetPrinter(createGreetInTurkish, "Hatice")
  greetPrinter(createGreetInEnglish, "Mary")
  greetPrinter(convertToUppercase, "naber?")

  greetCreator := createGreetInTurkish
  greetPrinter(greetCreator, "Zeynep")

  func(name string) {
     greeting := "Selam " + name + " :)"
     fmt.Printf("%s\n", greeting)
  }("Yesim")

  closure := func(name string) {
     greeting := "Selam " + name + " :)"
     fmt.Printf("%s\n", greeting)
  }
  closure("Fatma")
  anotherGreetPrinter(closure, "Zeynep")
}
...
```

```go
...
func createGreetInTurkish(name string) string {
  return "Selam " + name + " :)"
}

func createGreetInEnglish(name string) string {
  return "Hi " + name + " :)"
}

func convertToUppercase(arg string) string {
  return strings.ToUpper(arg)
}

func greetPrinter(function func(it string) string, name string){
  var greeting = function(name)
  fmt.Printf("%s\n", greeting)
}

func anotherGreetPrinter(function func(it string), name string){
  function(name)
}
```

greetPrinter bir hire order function olarak davranıyor kendisine bir function alıyo. O function ismi function ben o functiona function olarak ulaşacağım. Tanımı func(it string) string ona bir string veriyorum o stringi alıp bişeyler yapıyo bana string döndürüyor. Bu bir string alıp string döndüren func prototipi. Bir signiture. Onu geçiyoum birde argüman olarak name geçiyorum. O name'i alıp. functionunun parametresi it'tir. İsmi geçersem ismi alır it'e geçer. O geçilen fonksiyon o it'le ne yapıyorsa bana bir tane string döner.  Sarıyla iaşretlediğim yerin üstünde 3 function string alıp string dönüyo yani benim greetPrinter'ın bir örneği gibi düşünebilirsin.

```go
func(name string) {
    greeting := "Selam " + name + " :)"
    fmt.Printf("%s\n", greeting)
}("Yesim")
```

bu fonksiyona bakalım ismi yok bu gibi funclara anonymous functionlar denir. İsimsiz fonksiyonlar.

Anonymus funcları istersem tutup bir değişkene atayabilirim. Genelde anonuymus funclara clouser denir

```go
closure := func(name string) {
    greeting := "Selam " + name + " :)"
    fmt.Printf("%s\n", greeting)
}
closure("Fatma")
```

Clouser e Fatma geçince selam Fatma yazar.

Go blok yapılı bir dildir. Bloklar bracketlerla belirlenir. Go bracketlera çok önem verir nerede olup olmadığı bile çok önemlidir.

Go semicolon gerektirmez.

Go tamamen platform bağımlı bir dildir. Executable kodu alıp farklı platformlarda çalıştırmamız ya imkan dışıdır yada zordur soruce code'a değişiklikler yapabiliriz. Platformdan kasıt operative system ve hardware cpu. Farklı platformlarda tekrar tekrar complier etmemiz lazım.

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

herhangi bir entity bu bir varailable function structlar olabilir. İçinde bulunduğu pakette declare edilirken büyük harfle başlıyorsa isminin ilk harfi exporteddir dolayısıyla visilabledir. Değer publictir. Bu kuralla birlikte metotu getter yerine Büyük harf ile başaltırsak erişilebilir hale getiririz.

Paket isimlerinde olabildiğince kısa isimler. Olabildiğince camel case kullanalım.

Rest apiler

```go
package main

import (
  "net/http"

  "github.com/gorilla/mux"
)

func main() {
  r := mux.NewRouter()
  r.HandleFunc("/greet/{name}", greet)
              .Methods(http.MethodGet)
  http.ListenAndServe(":8080", r)
}

func greet(w http.ResponseWriter, r *http.Request) {
  w.Header().Set("Content-Type", "application/json")
  w.WriteHeader(http.StatusOK)
  vars := mux.Vars(r)
  var name = vars["name"]
  w.Write([]byte("Hello " + name + "!"))
}
```

{name} ile programatik olarak gelen değeri alıyorum.

vars[name] ilede direk içindeki değeri alabiliyorum.