



Golang Türkçe Eğitim Kaynağı

GİRİŞ

Bu eğitim kaynağında Golang programlama dili hakkında bilgilerdirici ön yazı, kullanım şekline ve örneklerine bakacağız. Bu kitapla pratik yapabilirsiniz. Kitap ileri seviye Go programlama içermeyecektir.

Bu kitap kimlere hitap ediyor?

Bu kitap;

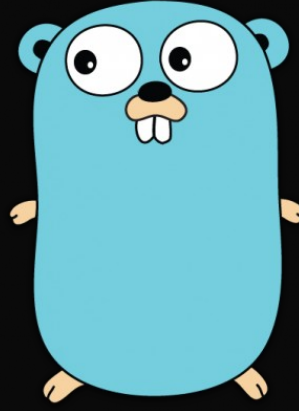
- Golang öğrenmek isteyen,
- Golang'ta giriş seviyesinde bilgi sahibi olan,
- Golang'ta orta seviye bilgi sahibi olan,
- Daha önceden başka diller kullanmış olan kişilere hitap ediyor.

Amaç

- Golang için Türkçe kaynak oluşturmak
- Golang için ücretsiz eğitim kaynağı oluşturmak
- Golang dilinin temel yapısını öğretmek

Kitap içeriği hakkında

Kitabın içeriğinde Go programlama dilinden “Go”, “Golang” ve “Go Programlama Dili” olarak bahsediyor olacağım. Hepsi aynı anlama geliyor. Genellikle kodların kullanım şekli ve yapısından bahsediyor olacağım. Tabi ki işleyişi anlayabilmemiz için örnekler ile pekiştireceğim.



Golang Nedir?

Golang (diğer adıyla Go), Google'ın 2007 yılından beri geliştirdiği açık kaynaklı programlama dilidir. Daha çok alt-sistem programlama için tasarlanmış olup, derlenmiş ve statik tipli bir dildir. İlk versiyonu Kasım 2009'da çıkmıştır. Derleyicisi olan “gc” (Go Compiler) açık kaynak olarak birçok işletim sistemi için geliştirilmiştir.

Golang, Google Mühendislerinden olan Robert Griesemer, Rob Pike ve Ken Thompson tarafından ilk olarak deney amaçlı ortaya çıkarılmıştır. Diğer dillerdeki eleştirilen sorunlara çözüm getirmek ve iyi yönlerini korumak amaçlı çıkarılmıştır.

Dilin özellikleri;

- Statik yazılmıştır, fakat dinamik rahatlığındadır.
- Büyük sistemlere ölçeklenebilir.
- Üretken ve okunabilir olması, çok fazla zorunlu anartar kelime ve tekrarlamaların kullanılmaması
- Tümüleşik Geliştirme Ortamına (IDE) ihtiyaç duymaması fakat desteklemesi
- Ağ ve çoklu işlemleri desteklemesi
- Değişken tanımında tür belirtimi isteğe bağlıdır.
- Hızlı derlenme süresi



Anahtar Kelimeler

Meraklısı için Golang'ta kullanılan tüm anahtar kelimeler (25);

break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

Bilgisayarınız üzerinde Golang geliştirme

Öncelikle bilgisayarımız üzerinde nasıl Golang geliştireceğimize bakalım. Geliştirmek derken Golang programı oluşturacağımızı kastediyorum. Öncelikle Golang'ın resmi sitesinden Golang programını indiriyoruz.

Buradan indirebilirsiniz

<https://golang.org/dl/>

Golang'ın basit bir kurulumu var o yüzden kurulumu atlıyorum. Linux İS kullananlara tavsiyem, kullandığınız dağıtımın uygulama deposundan Golang'ı indirin. Sizin için daha kolay olur.

Golang'ı indirdiğimize göre bize Golang kodlarımızı yazacağımız bir Tümüleşik Geliştirme Ortamı (IDE) lazım. IDE'ler kodlarımızı yazarken kodların doğruluğunu kontrol eder ve kod yazarken önerilerde bulunur. Bu da kod yazarken işimizi kolaylaştırır.

Benim tavsiyem çoğu kodlama dilini yazarken kullandığım ve Golang yazanların da popüler olarak kullandığı Visual Studio Code programı.

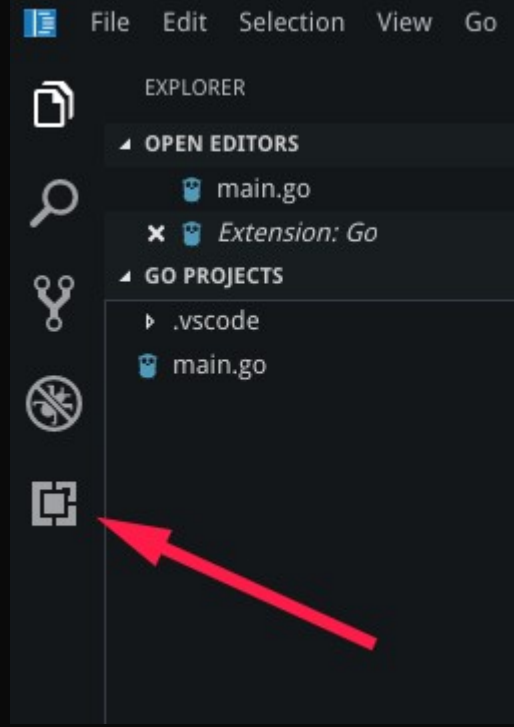
Buradan indirebilirsiniz

<https://code.visualstudio.com/Download>

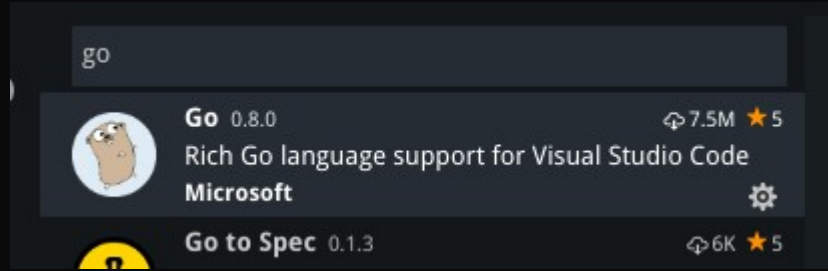
Linux İS kullananlara yine kullandıkları dağıtımın uygulama deposundan indirmelerini tavsiye ediyorum.

Visual Studio Code'dan ilerki zamanlarda **vscode** olarak bahsedeceğim.

Visual Studio Code'u kurduğumuza göre, vscode üzerinde Golang kodlarımızın kontrolü ve derlenmesi için Golang eklentisini kuralım. Visual Studio Code programında pencerenin sol tarafında eklentiler sekmesine giriyoruz.



Eklenti arama yerine “go” yazıyoruz ve resimde de göstereceğim eklentiye **Install** butonuna basarak yüklüyoruz. Ben de yüklü olduğu için gözüküyor



Eklenti yükledikten sonra vscode'u yeniden başlatın. Böylelikle vscode üzerinde Golang programları yazabiliyor olacağız. Eğer sorun yaşarsanız yazının devamını okuyabilirsiniz. Genellikle Windows kullananların ek bir işlem yapması gerekir.

Visual Studio Code'a Go eklentisini kurduktan sonra eklentinin çalışır durumda olması için Git yazılımının bilgisayarda kurulu olması gerekir. GNU/Linux işletim sisteminde yüksek ihtimal kurulu olarak gelir. Windows'larda varsayan olarak kurulu gelmez. O yüzden Git-bash programını indirmeniz gerekir. Bu sayede Visual Studio Code üzerinden Go eklentisinin kurulumunu başarıyla tamamlayabilirsiniz. Git-Bash'in kurulu olup olmadığını anlamak için aşağıdaki komutu deneyebilirsiniz.

```
git --version
```

Şöyle bir sonuç alırsanız kurulmuş demektir.

```
ksc10@ksc10-PC /home/ksc10 «system»  
$ git --version  
git version 2.19.1
```

Eğer Git-Bash'i yeni kurduysanız bilgisayarınızı yeniden başlatmayı unutmayın aksi takdirde Go eklentisinin kurulumunu tamamlayamazsınız.

Herşey hazır olduktan sonra oluşturduğunuz .go uzantılı dosyayı vscode üzerinde açın. Ekranı birkaç birşey yazdıktan sonra sağ altta Install All uyarısı çıkacak. Orada çıkan herşeyi yükledikten sonra eklentinin kurulumu başarıyla sonlanacak.

Artık vscode üzerinde sorunsuzca Go dili geliştirebiliriz.



BÖLÜM 1

- İlk Uygulama (Merhaba Dünya)
- Derlenme ve İnşa Etme (Compile & Build)
- Build Tablosu
- Paketler
- Yorum Satırı
- Veri Tipleri
- Değişkenler
- Sabitler
- Tür Dönüşümü
- Fonksiyonlar
- Aritmetik Operatörler
- İlişkisel Operatörler
- Atama Operatörleri

Merhaba Dünya

Programlama dünyasında gelenektir, bir programlama dili öğrenilirken ilk önce ekrana “Merhaba Dünya” çıktısı veren bir program yazılır. Biz de geleneği bozmadan Golang üzerinde Merhaba Dünya uygulaması yazalım. İlk önce kodları görelim. Daha sonra açıklamasını görelim.

```
package uygulama

import "fmt"

func main() {
    fmt.Println("Merhaba Dünya")
}
```

Şimdi yukarıdaki kodlarda neler yaptığımıza geleyim. Package, kod sayfalarımız arasında iletişimde bulunabilmemizi sağlar. Bu sayede içerisinde package değeri aynı olan kod sayfaları birbirleriyle iletişim halinde olma yeteneği kazanır. Yukarıdaki örnekte **package uygulama** olan sayfalar birbiriyle iletişim kurabilir.

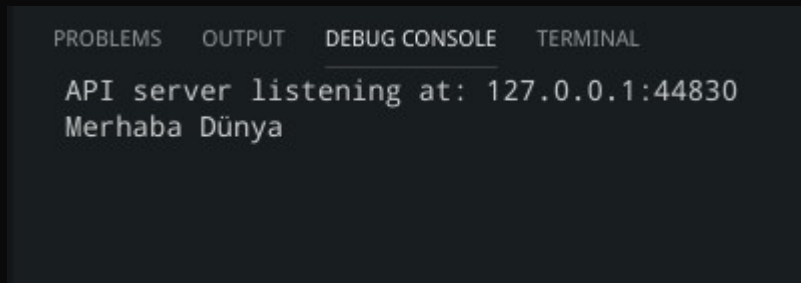
import “fmt” ile Golang dilinde kullanılan ana işlemler için olan kütüphanemizi içeri aktardık.

func main() ise programımızın çalışacağı ana bölümün fonksiyonudur. Yanındaki süslü parantezler **{ }** içine yazdığımız kodlar ile programımızda çeşitli işlemler yapabileceğiz. Derlenmiş bir uygulama ilk olarak main fonksiyonuna bakar ve buradaki kodları çalıştırır.

Yukarıda import ettiğimiz **fmt** kütüphanesi içinden **Println** fonksiyonu ile ekranımıza **“Merhaba Dünya”** yazısını bastırdık.

Gelelim programımızın derlenmesine. Daha önceden programlama dilleriyle geçmiş olmaya arkadaşlarımız için derlenme şöyle anlatılabilir. Yazdığımız Golang dili insanların kolaylıkla programlama yapabilmesi için oluşturulmuş bir dildir. Ama makine (bilgisayar) bu yazdıklarımızı anlamaz. O yüzden her derlenen dilde olduğu gibi Golang'ın da yazdıklarımızı makinenin anlayacağı makine diline çeviren derleyicisi vardır.

Makinemiz üzerinde çalıştırılabilir bir dosya üretmek için kodlarımızın derlenmesi gereklidir. Vscode üzerinden kodlarımızın derlenip çalışması için **F5** tuşuna basıyoruz. Böylece programımızı test edebiliriz. Eğer vscode üzerinden derliyorsanız yazdığımız kodların hemen altında **DEBUG CONSOLE** bölümünde kodumuzun sonuç çıktısını görebiliriz.

A screenshot of the VS Code interface showing the 'DEBUG CONSOLE' tab. The output text reads: 'API server listening at: 127.0.0.1:44830' followed by 'Merhaba Dünya' on the next line. The tabs at the top are 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'.

Çıktımızı inceleyecek olursak, **API server listening at :127.0.0.1:44830** ibaresinde gerçekleşen olay, Golang kodlarımızı çalıştırdığımızda oluşturulan **44830** portlu **127.0.0.1** yerel sunucusu (localhost) üzerinden kodlarımızın sürüş testini gerçekleştirdik. Hemen aşağısına da çıktımızı verdi.

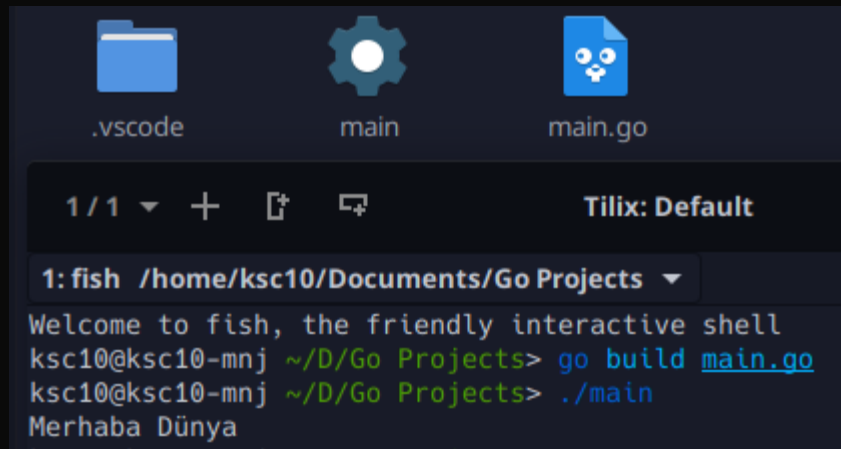
Eğer vscode üzerinden değil de, **konsol** üzerinden yapmak isterseniz, oluşturmuş olduğumuz main.go dosyasının bulunduğu dizin (klasör) içerisinde konsol uygulamamızı açıyoruz. Windows'ta **cmd** veya **Powershell**'dir. Unix sistemlerde **terminal** diye geçer. İki tarafa da yazacağımız komutlar aynıdır. O yüzden hangisinden yazdığınız farketmez.

Kodlarımızı sadece denemek istiyorsak yazacağımız komut `go run main.go` //main.go yerine .go dosyamızın ismi gelecek.

Eğer çalıştırılabilir bir dosya oluşturmak istiyorsak (Windows'ta .exe) `go build main.go` //main.go yerine .go dosyamızın ismi gelecek. Böylece bu işlemleri yaptığımız dizin içerisine çalıştırılabilir bir dosya oluşturmuş olduk.

Windows üzerinden konsola `main` yazarak uygulamayı açabilir veya klasörde `main.exe` dosyasına çift tıklayarak uygulamayı çalıştırabilirsiniz.

Linux üzerinden ise terminale derlenmiş main çıktınızın bulunduğu dizinde `./main` yazarak çalıştırabilirsiniz. Linux üzerinden şöyle bir sonuç elde etmiş olacaksınız.



The screenshot shows a terminal window with a dark background. At the top, there are three icons: a folder icon labeled '.vscode', a gear icon labeled 'main', and a document icon labeled 'main.go'. Below these icons is a toolbar with a dropdown menu showing '1 / 1', a plus sign, a magnifying glass, and a refresh icon. To the right of the toolbar is the text 'Tilix: Default'. Below the toolbar is a terminal prompt '1: fish /home/ksc10/Documents/Go Projects' followed by a dropdown arrow. The terminal output shows the following text: 'Welcome to fish, the friendly interactive shell', 'ksc10@ksc10-mnj ~/D/Go Projects> go build main.go', 'ksc10@ksc10-mnj ~/D/Go Projects> ./main', and 'Merhaba Dünya'.

Böylece ilk uygulamamızı yazmış olduk. Tabi şu ana kadar görmemiş olduğumuz kodlar gördük. Onların açıklamaları da ileriki bölümlerde olacak. Şimdilik gelenek diye ilk bölümde Merhaba Dünya uygulaması yazdık.

FARKLI PLATFORMLARA BUILD ETME

Golang projemizi build (inşa) ederken, yani çalıştırılabilir bir dosya üretirken **go build dosya.go** şeklinde build ederiz. Bu işlem varsayılan olarak kullanmakta olduğumuz işletim sistemi için build işlemi yapar. Yani Windows kullanıyorsak Windows'ta çalışmak üzere Linux İS kullanıyorsak Linux İS'te çalışmak üzere dosya oluşturur. Aynı şekilde sistemimizin mimarisi 32bit ise 32bit için, 64bit ise 64bit için çalıştırılabilir dosya üretir. Örnek olarak sistemimiz Windows 64bit ise oluşturduğumuz çalıştırılabilir dosya (exe dosyası) sadece Windows 64bitlerde çalışır.

Eğer farklı işletim sistemi için bir çalıştırılabilir dosya üretmek istiyorsak aşağıdaki komutu kullanmamız gerekir.

```
env GOOS=hedef-sistem GOARCH=hedef-mimari go build hedef-dosya
```

Örnek olarak, **main.go** dosyamızı **Windows 32bit** sistemler için build etmek istersek aşağıdaki komutları girmemiz gerekir.

```
env GOOS=windows GOARCH=386 go build main.go
```

Bu işlem ile **main.exe** adında bir dosya elde ederiz. Bu dosya Windows 32bit sistemlerde çalışabilir. Biliyorsunuz ki 32bit uygulamalar 64bit sistemlerde çalışır ;fakat 64bit uygulamalar 32bit sistemlerde çalışmaz. Onun için bir uygulama build ediyorken bunu aklınızdan çıkarmayın. Golang'ın dosya build edebildiği işletim sistemi ve mimari sayısı oldukça fazladır.

Golang'ın build edebildiği tüm işletim sistemi ve mimarilerine bakmak gerekir ise;

GOLANG BUILD TABLOSU

GOOS	GOARCH
android	arm
darwin	386
darwin	amd64
darwin	arm
darwin	arm64
dragonfly	amd64
freebsd	386
freebsd	amd64
freebsd	arm
linux	386
linux	amd64
linux	arm
linux	arm64
linux	ppc64
linux	ppc64le
linux	mips
linux	mipsle
linux	mips64
linux	mips64le
netbsd	386
netbsd	amd64
netbsd	arm
openbsd	386
openbsd	amd64

GOOS	GOARCH
openbsd	arm
plan9	386
plan9	amd64
solaris	amd64
windows	386
windows	amd64

Tablo harf sıralamasına göre yapılmıştır.
Birkaç örnek daha yapmak gerekirse;

Windows 64bit build

```
env GOOS=windows  
GOARCH=amd64 go build  
main.go
```

Linux 32bit build

```
env GOOS=linux  
GOARCH=386 go build  
main.go
```

macOS 64bit build

```
env GOOS=darwin  
GOARCH=amd64 go build  
main.go
```

Paketler

Her Golang programı paketlerden oluşur ve kendisi de bir pakettir.

```
package uygulama

import "fmt"

func main() {
    fmt.Println("Merhaba Dünya") // Çıktımız
}
```

package terimi ile programımızın paket adını belirleriz. Hemen aşağısında da **import "fmt"** ile fmt paketini çektiğimizi görebilirsiniz. Yani çektiğimiz **fmt paketi** de bir programdır. Bilin bakalım fmt paketinin ilk satırında ne yazıyor? Tabiki de **package fmt**. Yani package ile programımızın ismini tanımlıyoruz. Bu ismi kullanarak diğer paketler ile iletişimde bulunabiliriz.

import terimi ise yazıldığı pakete başka bir paketten bir yetenek aktarmaya yarar. Yetenekten kastım, import edilen paketin içinde fonksiyonlar mı var? Struct'lar mı var? vs. onları içeri aktarır.

```
import (
    "fmt"
    "math"
)
```

Yukarıda birden fazla paket import etmeyi görüyoruz. **math** paketi bize ileri matematiksel işlemler yapabilmemiz için gerekli fonksiyonları sağlar.

Yorum Satırı

Diğer dillerde de olduğu gibi Golang'ta yorum satırı özelliği mevcuttur. Yorum satırı derleyici tarafından işlenmez. Yani görmezden gelinir. Bu bölüme kendiniz için açıklama vs. bilgiler yazabilirsiniz. Golang'ta yorum satırı oluşturmak için 2 yöntem mevcuttur.

// Çift Taksim Yöntemi

Bu yöntem ile derlenmesini istemediğimiz yazının başına çift taksim ekleyerek görmezden gelinmesini sağlıyoruz.

```
func main() {  
    var isim = "Ali" //Buraya açıklama gelecek  
    fmt.Println(isim)  
    fmt.Println(math.Pi)  
}
```

/* */ Taksim-Yıldız Yöntemi

Bu yöntem ile birden fazla satırın derlemede görmezden gelinmesini sağlayabiliriz.

```
func main() {  
    var isim = "Ali"  
    /* Birden  
    Fazla  
    Satır */  
    fmt.Println(isim)  
    fmt.Println(math.Pi)  
}
```


Veri Tipleri

Her programlama dilinde olduğu gibi Golang'ta da veri tipleri mevcuttur. Veri tipleri verilerimizi RAM'de saklayabilmemiz için belirttiğimiz şekilde alan ayırmamızı sağlar. Veri tiplerini inceleyecek olursak;

SAYISAL VERİ TİPLERİ

Integer Türler

Öncelikle tüm integer türleri bir görelim;

`int, int8, int16, int32, int64`

`uint, uint8, uint16, uint32, uint64, uintptr`

Bu veri tipleri içerisinde sayısal değerleri depolayabiliriz. Fakat şunu da unutmamalıyız. Her sayısal veri tipinin depolayabildiği maksimum bit vardır. Örnek olarak `int8` veri tipinin maksimum uzunluğu 8 bit'tir. Bitler `0` ve `1` sayılarından oluşur. 8 bit demek 8 haneli 1 ve 0 sayısı demektir. `int8` maksimum alabileceği sayı derken `11111111` (8 tane 1), yani onluk sistemde 255 sayısına denk gelir. Pozitif olarak `+127`, negatif olarak `-128` maksimum değerinin alabilir. ($127+128=255$). `int16` `+32767` ve `-32768` maksimum değerlerini alır. `int32` `+2147483647` ve `-2147483648` maksimum değerlerini alır. `int64` `+9223372036854775807` ve `-9223372036854775808` maksimum değerini alır.

`U` harfi ile başlayan sayı veritiplerinde ise sayının değeri pozitif veya negatif işaretle değildir. Sadece bir sayısal değerdir. `U`'nun anlamı `unassigned` yani işaretsizdir. `uint8` `0-255` arası, `uint16` `0-65535`, `uint32` `0-4294967295` arası, `uint64` `0-18446744073709551615` arası değerler alabilir. `uintptr` ise yazdığınız sayıya göre alanı belirlenir.

Integer sayısal veri tipleri içerisinde bahsedebileceğimiz son tipler ise **int** ve **uint**. Int ve uint veri tipleri kullanmış olduğumuz işletim sistemi 32bit ise 32bit değer alırlar, 64bit ise 64bit değer alırlar. Sayısal bir değer atanacağı zaman en çok kullanılan veri tipleridir. Genellikle **int** daha çok kullanılır. Eğer çok meşakkatli bir program yazmayacaksanız **int** kullanmanız önerilir.

Byte Veri Tipi: **uint8** ile aynıdır.

Rune: **int32** ile aynıdır. Unicode karakter kodlarını ifade eder.

Float Türler

Float türleri integer türlerden farklı olarak küsürlü sayıları tutar. Örnek: 3.14

Lütfen Dikkat!

Küsürlü sayılar İngiliz-Amerikan sayı sistemine göre nokta koyarak ifade edilir. Türk sistemindeki gibi virgül (3,14) ile ifade edilmez.

float32: 32bitlik değer alabilir.

float64: 64 değer alabilir.

Complex Türler

Complex türleri içerisinde gerçel küsürlü (float) ve sanal sayılar barındırabilir. Türkçe'de karmaşık sayılar diye adlandırılır.

complex64: Gerçel float32 ve sanal sayı değeri barındırır.

complex128: Gerçel float64 ve sanal sayı değeri barındırır.

Sayısal türler bu şekildedir.

BOOLEAN VERİ TİPİ

Boolean yani mantıksal veri tipi bir durumun var olması halinde olumlu (true) değer, var olmaması halinde olumsuz (false) değer alan veri tipidir.

STRING VERİ TİPİ

String yani dizgi veri tipi içerisinde metinsel ifadeler barındırır. Örnek olarak “Golang çok güzel ama ingilicce”. String veri tipi değeri çift tırnak (“Değer”) içine yazılır. Diğer dillerdeki gibi tek tırnak (‘Değer’) insiyatifi yoktur. Tek tırnakla kullanım başka bir amaç içindir. İlerde onu da göstereceğim.

Özet olarak Veri Tipleri

Veri tipleri atanacak değerlerimizi RAM üzerinde depolamak için kullandığımız araçlardır. Tam sayı değerler için Integer veri tiplerini, ondalık sayılar için Float veri tiplerini, mantıksal değerler için Boolean veri tipini, metinsel değerler için String veri tipini kullanırız. Karmaşık sayı değerleri için ise Complex veri tipini kullanırız.

"Türkiye"	= String Tipi
1881	= Integer Tipi
10,5	= Float Tipi
True	= Boolean Tipi
2+3i	= Complex Tipi

Veri Tiplerinin Varsayılan Değerleri

Veri tipleri içerisine değer atanmadan oluşturulduğu zaman varsayılan bir değer alır.

- Sayısal Tipler için 0,
- Boolean Tipi için false,
- String Tipi için "" (Boş dizgi) değeri alır.

Aritmetik Operatörler

Aritmetik operatörler programlamada matematiksel işlemler yapabilmemize olanak sağlar.

Operatör	Açıklama	Örnek
+	Toplar	$2+5=7$
-	Çıkarır	$10-3=7$
*	Çarpar	$3*4=12$
/	Böler	$10/2=5$
%	Bölümden kalanı verir(Mod).	$10\%3=1$
++	1 arttırır	$1++=2$
--	1 eksiltir	$3--=2$

İlişkisel Operatörler

İlişkisel operatörler programlamada iki veriyi birbiriyle karşılaştırabilmemize olanak sağlar. Karşılaştırma doğrulanıyorsa **true** değer, doğrulanmazsa **false** değer alır.

Operatör	Açıklama	Örnek
==	İki verinin eşitse true verir	2==3 (false)
!=	İki verinin eşit değilse true verir	2!=3 (true)
>	1. veri 2. veriden büyükse true verir	5>3 (true)
<	1. veri 2. veriden küçükse true verir	4<6 (true)
>=	1. veri 2. veriden büyük veya eşitse true verir.	4>=3 (true) 4>=4 (true)
<=	1. veri 2. veriden küçük veya eşitse true verir	3<=2 (false) 3<=3 (true)

Mantıksal Operatörler

Mantıksal operatörler birden fazla mantıksal veriyi kontrol eder ve kullandığımız operatöre göre mantıksal değer döndürür.

Operatör	Açıklama	Örnek
&&	VE operatörüdür. 2 değer de true ise true değer döndürür.	2==2 && 2==3 (false)
	VEYA operatörüdür. 2 değerden biri true ise true değer döndürür.	2==2 2==3 (true)
!	DEĞİL operatörüdür. 2 değer mantıksal olarak karşılaştırıldığında çıkan sonucun tersini alır.	!(2==2&&3==3) (false)

Atama Operatörleri

Atama operatörleri değişkenlere ve sabitlere değer atamak için kullanılır. Aşağıdaki tabloda c'nin değeri 10'dur. (c=10)

Operatör	Açıklama	Örnek
=	Atama operatörüdür	c=2
+=	Kendiyle toplar	c+=2 (c=12)
-=	Kendinden çıkarır	c-=2 (c=8)
=	Kendiyle çarpar	c=2 (c=20)
/=	Kendine böler	c/=2 (c=5)
%=	Kendine bölümünden kalanı atar	c%=3 (c=1)

DEĞİŞKENLER VE ATANMASI

Değişkenler içerisinde değer barındırarak RAM'e kaydettiğimiz bilgilerdir. Değişkenler programımızın işleyişinde önemli bir role sahiptir. Değişkenleri şu şekillerde atayabiliriz. Değişkenler **var** ifadesi ile atanır. Tabi ki zorunlu değildir.

```
var isim string = "Ali"  
var yas int = 20  
var ogrenci boolean = true
```

Yukarıdaki yazdıklarımızı inceleyecek olursak;

var ile değişken atadığımızı belirtiyoruz. **isim** diye bir değişken adı atadık ve içine **"Ali"** değerinde **string** tipinde bir değer yerleştirdik. String tipi değerler çift tırnak içine yazılır.

Aynı şekilde **yas** adında değişken oluşturduk. **yas** değişkeni içerisine **int** tipinde **20** değerini yerleştirdik

Son olarak **ogrenci** adında bir değişken oluşturduk ve **true** değerinde **boolean** tipinde bir atama yaptık.

Golang'ta değişken adı oluştururken Türkçe karakterler kullanabiliriz. Örnek olarak **ogrenci** yerine **öğrenci** yazabilirdik. Ama başka bir programlama diline geçtiğinizde Türkçe harf desteklememesi halinde alışkanlıklarınızı değiştirmeniz gerekecek. O yüzden Türkçe karakter kullanmamanızı tavsiye ederim. Tabi ki zorunlu değil.

Programlama dillerinde, matematiğin aksine **=** (**eşittir**) işareti **eşitlik** için değil, **atamalar** için kullanılır.

Değişkenlerin atanması için farklı yöntemler de var. Diğer yöntemlere değinmek gerekirse;

Değişken atamasında illaki değişkenin veri tipini belirtmemiz gerekmez. Yazdığımız değere göre Golang otomatik olarak veri tipini algılar.

```
var isim = "Ali"  
var yas = 20  
var ogrenci = true
```

isim değişkeninin değerini çift tırnak arasına yazdığımız için **string** veri tipinde olduğunu algılayacaktır.

yas değişkeninin değerini sayı olarak girdiğimiz için **int** tipinde olduğunu algılar. Eğer **20** değil de **2.12312** gibi küsürlü bir değer girseydik veri tipini **float** olarak algılardı.

ogrenci değişkeninin değerini mantıksal olarak girdiğimiz için **boolean** veri tipinde olduğunu algılayacaktır.

En basit şekilde değişken ataması yapmak istersek;

```
isim:="Ali"  
yas:=20  
ogrenci:=true
```

Başına var eklemekten de değişken atamak mümkündür. Bu şekilde yapmak için **:=** işaretlerini kullanırız. Aynı şekilde bu yöntemde de verinin tipi otomatik algılanır.

Eğer değişken tanımlar iken değer kısmını boş bırakırsak yani; **var yas int** şeklinde yazarsak, önceki konuda da bahsettiğimiz gibi varsayılan olarak **0** değerini alır.

SABİTLER

Sabitler de değişkenler gibi değer alır. Fakat adından da anlaşılacağı üzere verilen değer daha sonradan değiştirilemez.

Sabitler tanımlanırken başına **const** eklenir. Örnek olarak;

```
const isim string = "Ali"  
const isim = "Ali"  
const yas = 20
```

const ile **:=** bereber kullanılamaz.

Yanlış kullanım: **const isim := "Ali"**

Doğru kullanım: **const isim = "Ali"**

Örnek olarak bir sabitin değerini atandıktan sonra değiştirmeye çalışalım. Aranızda ne olacağını merak eden çılgınlar olabilir.

```
const isim string = "Ali"  
isim="Veli"
```

Bu şekilde yazıp kodlarımızı derlediğimizde hata almamız kaçınmaz. Derlediğimizde **cannot assign to isim** hatasını verecektir. Yani diyor ki isim'e atanamıyor.

TÜR DÖNÜŞÜMÜ

Tür dönüşümü şu şekilde gerçekleştirilir.

`tür(değer)`

Örnek olarak bakmak gerekir ise;

```
i := 42  
f := float64(i)  
u := uint(f)
```

Yukarıdaki yapılan işlemleri açıklayacak olursak eğer, `i` adında bir `int` değişken tanımladık. `f` adındaki değışkende `i` değışkenini `float64` türüne dönüştürdük. `u` adındaki değışkende ise `f` değışkenini `uint` türüne çevirdik.

Tüm türler arasında bu şekilde dönüşüm gerçekleştiremezsiniz. Bir sayıyı `string` tipine dönüştürmek istediğimizde ne olacağına bakalım.

```
deneme:=string(8378)  
fmt.Println(deneme)
```

`deneme` adındaki değerimizin içinde `8378` sayısını `string` türüne dönüştürdük ve hemen aşağısına `deneme`'nin aldığı değeri ekrana bastırması için kodumuzu yazdık.

Aldığımız konsol çıktısı şu şekilde olacaktır.

₺

Yani Türk Lirası simgesi çıkacaktır. Sayılar `string` türüne dönüştürüldüğünde karakter olarak değeri alır.

FONKSİYONLAR

Fonksiyonlar içlerine parametre girilebilen ve işlemler yapabilen birimlerdir. Matematikteki fonksiyonlar ile aynı mantıkta çalışan bu birimlerden bir örneği inceleyelim.

```
package main

import "fmt"

func topla(a int, b int) int {
    return a + b //a ve b'nin toplamını döndürür.
}

func main() {

    fmt.Println(topla(2, 5)) //2+5 sonucunu ekrana bastır
}
```

Yukarıdaki kodları ineleyecek olursak, fonskiyonlarımızı oluşturmak için **func** anahtar kelimesini kullanırız. Yanına ise fonskiyonumuzun ismini yazarız. Parantez içine fonksiyonumuzun dışarıdan alacağı parametreler için değişken-tip tanımlaması yaparız. parantezin sağına ise fonksiyonun döndüreceği **return** değerinin tipini yazarız. Süslü parantezler içinde fonsiyonumuzun işlemleri bulunur. Son olarak **return** ile veri tipini belirlediğimiz değeri elde etmiş oluruz.

Main fonksiyonu içerisinde **topla(2,5)** fonksiyonu ile **2** ve **5** sayısının toplamını ekrana bastırmış olduk. Yani ekrana 7 sayısı verildi.

Fonksiyonlar istendiği kadar parametre alabildiği gibi, istenirse parametresizde olabilir. Fonksiyonları veri **return** etmek yerine bir işlem yaptırmak içinde kullanabiliriz.

```
package main

import "fmt"

func yazdir() {
    fmt.Println("yazı yazdırdık")
}

func main() {

    yazdir()
}
```

yazdir adlı fonsiyonumuzun parantezine değişken tanımlamadık ve parantezin sağına fonksiyon bloğu içerisinde **return** olmadığı için veri çıkış tipini belirtmedik. Fonksiyonumuzun içerisinde sadece ekrana yazı bastırdık.

Fonksiyonlar Hakkında Ayrıntılı Bilgiler

Fonksiyon parantezi içerisine değişken tanımlanırken eğer tüm değişkenlerin türleri aynı ise sadece en sağdaki değişkenin tipini belirtmeniz yeterlidir. Örnek:

```
func islem(a, b, c, d int) int {
```

Fonksiyonlar birden fazla sonuç verebilir. Bunu yapmak için;

```
package main

import "fmt"

func takas(a, b string) (string, string) {
    return b, a
}

func main() {
    fmt.Println(takas("Türkiye", "Cumhuriyeti"))
}
```

Çıktımız **Cumhuriyeti Türkiye** olacaktır.

```
package main

import "fmt"

func islem(sayi int) (x, y int) { //return'un degiskenlerini tanımladık
    x = sayi / 2
    y = sayi * 2
    return //Burada sadece return yazıyor
}

func main() {
    fmt.Println(islem(10))
}
```

Yukarıda ise isimlendirilmiş **return** kullandık. **return** tipini yazdığımız paranteze bakacak olursa **(x, y int)** diyerek **return** edilecek verinin fonksiyonun blokları içerisinde çekilmesini sağladık. Böylece fonksiyon bloğunun sonundaki **return** kelimesinin yanına birşey yazmadık. Bu fonksiyonumuzun çıktısı ise **5 20** olacaktır.



BÖLÜM 2

- Döngüler
- If-Else
- Switch
- Defer

DÖNGÜLER

Programlama ile uğraşan arkadaşlarımızın da bileceği üzere, programlama dillerinde **while**, **do while** ve **for** döngüleri vardır. Bu döngüler ile yapacağımız işlemin belirli koşullarda tekrarlanmasını sağlayabiliriz. Golang'ta ise diğer dillerin aksine sadece **for** döngüsü vardır. Ama bu **while** ve **do while** ile yapılanları yapamayacağımız anlamına gelmiyor. Golang'taki **for** döngüsü ile hepsini yapabiliriz. Yani dilin yapımcıları tek döngü komutu ile hepsini yapabilmemize olanak sağlamışlar.

Gelelim **for** döngüsünün kullanımına. Go'da **for** döngüsü parametreleri parantez içine alınmaz.

STANDART FOR KULLANIMI

```
for i:=0; i <10; i++ {  
    fmt.Println(i)  
}
```

Açıklaması:

Döngü değişkenimiz olan **i**'ye **0** sayısal değerini verdik. **i<10** yazmamızın sebebi alt bloktaki kodun sadece **i** değeri **10** sayısal değerinden **küçük** olduğu zaman çalışmasını sağladık. **i++** ile ise döngü her başa sardığında **i**'ye **+1** sayı eklemesini sağladık. **for** kod bloğunun içinde ise her işlemde konsola **i**'nin değerinin bastırılmasını sağladık.

Konsol çıktımız şu şekilde olacaktır;

```
0  
1  
2  
3  
4  
5
```



```
6  
7  
8  
9
```

SADECE KOŞUL BELİRTEREK KULLANMA

Bu **for** yazım şekli **while** mantığı gibi çalışır. Parametrelerde sadece koşul belirtilir.

```
deger:=0  
for deger <10 {  
    fmt.Println(deger)  
    deger++  
}
```

Açıklaması:

For döngüsünden ayrı olarak **deger** adında **0** sayısal değerini alan bir değişken oluşturduk. For döngüsünde ise **sadece koşul parametresini** belirttik. Yani döngü **deger** değişkeni **10** sayısından küçük olduğu zaman çalışacak. For kod bloğu içerisinde her döngü tekrarlandığında **deger** değişkeni ekrana basılacak ve **deger** değişkenine **+1** eklenecek.

Konsol çıktımız şu şekilde olacaktır;

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

IF-ELSE AKIŞI

If ve Else kelimelerinin Türkçe karşılığına bakacak olursak; If : Eğer, Else : Yoksa anlamına gelir. If-Else akışı koşullandırmalar için kullanılır. Diğer dillerin aksine koşul parametresi parantezler içine yazılmaz. Teorik kısmı bırakıp uygulama kısmına geçelim ki daha anlaşılır olsun :)

```
if koşul {  
    Koşul sağlandığında yapılacak işlemler  
} else {  
    Koşul sağlanmadığında yapılacak işlemler  
}
```

Yukarıdaki kod tanımına göre örnek bir program yazalım;

```
package main  
  
import "fmt"  
  
func main() {  
    i := 5  
  
    if i == 5 {  
        fmt.Println("i'nin değeri 5'tir.")  
    } else {  
        fmt.Println("i'nin değeri 5 değildir.")  
    }  
}
```

Yukarıdaki kodları inceleyelim. *i*'nin değerini 5 verdik. *if* teriminin sağında *i'nin 5 eşitliği* koşulunu sorguladık. Eşitse ekrana *i'nin değeri 5'tir.* yazısını bastırarak. Değilse *i'nin değeri 5 değildir.* yazısı bastırarak. *i*'nin değeri 5 olduğu için ekrana *i'nin değeri 5'tir.* yazısını bastırdı.

If-Else akışında else kullanmamız else'nin kod bloğunu boş bırakmamız ile aynı anlama gelir.

```
i:=10

if i==10 {
    fmt.Println("i'nin değeri 10'dur.")
}
```

Yukarıda sadece if deyimini girdik. Else'yi girmek. Burada sonuçlanan olay, i'nin değeri 10'a eşitse i'nin değeri 10'dur. yazısını ekrana bastırır. Else deyimini girmediğimiz için şartın sağlanmaması durumunda hiçbir işlem gerçekleşmez. Çıktımız i'nin değeri 10'a eşit olduğu için i'nin değeri 10'dur. çıkar.

ELSE-IF KULLANIMI

If-Else akışında birden fazla koşul kontrolü ekleyebiliriz. Bunu else if deyimi ile yapabiliriz. Kısaca bakacak olursak;

```
i := 5

if i == 5 {
    fmt.Println("i'nin değeri 5'tir.")
} else if i==3{
    fmt.Println("i'nin değeri 3'tür.")
} else{
    fmt.Println("i'nin değeri belirsiz.")
}
```

else if deyiminin yazılışını da gördük. Açıklamaya gelirse, else if deyimi kendinden önceki deyimin koşulunun sağlanmaması halinde bir sonraki koşulu kontrol ettirir. If-Else akışında istenildiği kadar else if deyimi eklenebilir.

Koşullar İçerisinde Operatör Kullanımı

Koşullar içerisinden mantıksal ve ilişkisel operatörler kullanılabilir. Operatörleri BÖLÜM 1'de görmüştük. Operatör kullanarak örnekler yapalım.

```
package main

import "fmt"

func main() {
    i := 5
    a := 3
    b := 5

    if i != a { //Birinci Koşul
        fmt.Println("i eşit değildir a")
    }

    if i == b { //İkinci Koşul
        fmt.Println("i eşittir b")
    }

    if i == b && i > a { //Üçüncü Koşul
        fmt.Println("i eşittir b ve i büyüktür a")
    }
}
```

Çıktımız şu şekilde olacaktır;

```
i eşit değildir a
i eşittir b
i eşittir b ve i büyüktür a
```

SWITCH AKIŞI

Switch kelimesinin Türkçe'deki anlamı anahtardır. Switch deyimi de **if-else** deyimi gibi koşul üzerine çalışır. Yine teorik kısmı geçip anlaşılır olması için örnek yapalım. **case** deyimi durumu ifade eder. Koşul sağlandığı zaman işleme devam edilmez.

```
package main

import "fmt"

func main() {
    i := 5

    switch i {
    case 5:
        fmt.Println("i eşittir 5")
    case 10:
        fmt.Println("i eşittir 10")
    case 15:
        fmt.Println("i eşittir 15")
    }
}
```

Çıktımız şu şekilde olacaktır;

```
i eşittir 5
```

Switch'te koşulların gerçekleşmediği zaman işlem uygulamak istiyorsak bunu default terimi ile yaparız. Örnek;

```
i := 5

switch i {
case 5:
    fmt.Println("i eşittir 5")
default:
    fmt.Println("i bilinmiyor")
}
```


Koşulsuz Switch

Switch'in tanımını daha iyi anlayabilmeniz için koşulsuz switch kullanımına örnek verelim. Bu yöntemde switch deyiminin yanına koşul girmek yerine case deyiminin yanına koşul giriyoruz.

```
package main

import "fmt"

func main() {
    i := 5

    switch {
    case i == 5: //i=5 olduğu için diğer case'ler sorgulanmaz
        fmt.Println("i eşittir 5")
    case i < 10:
        fmt.Println("i küçüktür 10")
    case i > 3:
        fmt.Println("i büyüktür 3")
    }
}
```

Çıktımız şu şekilde olacaktır;

```
i eşittir 5
```

DEFER

Defer kelimesinin Türkçe'deki karşılığı ertelemektir. Bu deyim yapacağımız işlemin başına eklersek o işlemi içerisinde bulunduğu fonksiyonun içindeki işlemlerden sonra çalıştırır. Çok karışık bir cümle kurdum ama uygulamaya geçince anlayacaksınız :)

```
package main

import "fmt"

func main() {
    defer fmt.Println("İlk Cümle")
    fmt.Println("İkinci Cümle")
}
```

Çıktımız şu şekilde olacaktır;

```
İkinci Cümle
İlk Cümle
```

Açıklamaya gelirsek ekrana İlk Cümle yazını bastırın satırımızın başına **defer** terimini ekledik. **defer** eklediğimiz satır **main()** fonksiyonunun içinde olduğu için **main()** fonksiyonundaki tüm işlemler tamamlandıktan sonra ekrana yazımızı bastırdı.

Birden fazla defer ekleyecek olursak;

```
package main

import "fmt"

func main() {
    defer fmt.Println("İlk Cümle")
    defer fmt.Println("İkinci Cümle")
    defer fmt.Println("Üçüncü Cümle")
    defer fmt.Println("Dördüncü Cümle")
    fmt.Println("Beşinci Cümle")
}
```

Çıktımız şu şekilde olacaktır;

```
Beşinci Cümle  
Dördüncü Cümle  
Üçüncü Cümle  
İkinci Cümle  
ilk Cümle
```

Burdan anlıyoruz ki en baştaki **defer** eklenen satır en son işleme tabi tutuluyor.

Hadi defer ile alakalı bir programlama alıştırması yapalım.

```
package main  
  
import "fmt"  
  
func main() {  
    fmt.Println("Sayıyor")  
  
    for i := 0; i < 10; i++ {  
        defer fmt.Println(i)  
    }  
  
    fmt.Println("Bitti")  
}
```

Çıktımız şu şekilde olacaktır;

```
Sayıyor  
Bitti  
9  
8  
7  
6  
5  
4  
3  
2  
1  
0
```



BÖLÜM 3

- Metodlar
- İşaretçiler (Pointers)
- Diziler (Arrays)
- Dilimler (Slices)
- Dilim Uzunluğu ve Kapasitesi
- Boş Dilimler
- Make ile Dilim Oluşturma
- Dilime Ekleme Yapma
- Range
- Map
- Arayüz

METODLAR

Golang'ta sınıflar yoktur. Ancak, türler üzerinden metod tanımlayabiliriz. Uzatmadan örneğimize geçelim.

```
package main

import "fmt"

type insan struct {
    isim string
    yas  int
    kilo int
}

func main() {
    ali := insan{}
    ali.isim = "Ali"
    ali.yas = 20
    ali.kilo = 70

    fmt.Println(ali.isim, ali.yas, ali.kilo)
}
```

Şimdi biz yukarıda ne yaptık?

insan tipinde bir struct ürettik ve bu struct içine **isim**, **yas** ve **kilo** isminde değişkenler atadık. Böylelikle programımıza yeni bir tür kazandırdık.

main() fonksiyonumuzun içinde ise, **ali** isminde **insan** dizisi oluşturduk. Böylece **ali** isimli nesnemiz insan türündeki tüm özelliklerden faydalabilir oldu. Hemen aşağısında ise **ali**'nin **isim**, **yas** ve **kilo** değerlerini atadık.

Daha sonra **ali** kişinin ismini, yaşını ve kilosunu ekrana bastırdık. Bu yöntemle diğer bir tabir ile **struct metodlar** denir

Çıktımız ise şöyle olacaktır;

```
Ali 20 70
```

Struct'ın mantığını anlamamız için struct yerine başka bir tip barındıran bir örnek yapalım,

```
package main

import "fmt"

type tamsayi int

func main() {
    var sayi tamsayi = 12
    fmt.Println(sayi)
}
```

Bu sefer tipi belirlerken **struct** yerine **int** tipini yazdık. Bu demek oluyor ki içerisinde **int** gibi tamsayı değer tutabilen **tamsayi** adında bir tür oluşturduk.

main() fonksiyonumuz içerisinden de görebileceğiniz üzere aynı bir değişken ataması yapar gibi **sayi** isminde **tamsayi** tipinde içerisindeki değer **12** olan bir değişken tanımladık ve bunu ekrana bastırdık. Çıktımız ise tahmin edebileceğiniz üzere **12** olacaktır.

İŞARETÇİLER

İşaretçiler yani pointer'lar bir değerin bellekteki adresini tutar. Değişken atamalarında & (and) işareti değişkenin bellekteki adresini tutar. * (yıldız) işareti ise tutulan adresteki değeri görüntüler. Tekrardan teorik kısmı kısa tutup örneğimize geçelim.

```
package main

import "fmt"

func main() {
    i := 40
    p := &i
    fmt.Println(p) //Alacağımız benzeri çıktı: 0xc000012120
    fmt.Println(*p) //Alacağımız çıktı: 40
    *p = 35 //Dolaylı olarak i nin değerini değiştirdik
    fmt.Println(i) //Alacağımız çıktı: 35
}
```

İşaretçilerin ana görevini anlatmak gerekir ise, işaretçiler yeni bir değişken oluşturmak yerine var olan bir değişkeni işaretler ve bu değişken üzerinde işlemler yapar. Kodlar ile değişiklikler yaparak mantığını kafanızda pekiştirebilirsiniz.

DİZİLER (Arrays)

Diziler içlerinde bir veya birden fazla değer tutabilen birimlerdir. Bir dizideki her değer sırasıyla numaralandırılır. Numaralandırma sıfırdan başlar. Aynı şekilde örneğe geçelim.

```
package main

import "fmt"

func main() {
    var a [3]string
    a[0] = "Ayşe" //Birinci değer
    a[1] = "Fatma" //ikinci değer
    a[2] = "Hayriye" //Üçüncü değer
    fmt.Println(a) //Çıktımız: [Ayşe Fatma Hayriye]
    fmt.Println(a[1])//Çıktımız: Fatma
}
```

Gelelim kodlarımızın açıklamasına. **a** isminde içerisinde 3 tane **string** tipinde değer barındırabilen bir dizi oluşturduk. **a** dizisinin birinci değerine yani **0** indeksine “**Ayşe**” atadık. **1** ve **2** indeksine ise “**Fatma**” ve “**Hayriye**” değerlerini atadık. **a** dizisini ekrana bastırdığımızda köşeli parantezler içinde dizinin içeriğini gördük. **a**’nın 1 indeksindeki değeri bastırdığımızda ise sadece 1 indeksindeki değeri gördük. Dizinin değerlerini tek tek olarak atayabileceğimiz gibi diziyi tanımlarken de değişkenlerini atayabiliriz.

```
package main

import "fmt"

func main() {
    a := [3]string{"Ayşe", "Fatma", "Hayriye"}
    fmt.Println(a) //Çıktımız: [Ayşe Fatma Hayriye]
}
```


DİLİMLER (Slices)

Dilimler bir dizideki değerlerin istediğimiz bölümünü kullanmamıza yarar. Yani diziyi bir pasta olarak düşünersek kestiğimiz dilimi yiyoruz sadece. Örneğimize geçelim.

```
package main

import "fmt"

func main() {
    a := [6]int{2, 3, 5, 6, 7, 9}
    fmt.Println(a) //Çıktımız: [2 3 5 6 7 9]
    var b []int = a[2:4] //Dilimleme işlemi
    fmt.Println(b) //Çıktımız: [5 6]
}
```

İnceleme kısmına geçelim. **a** isminde 6 tane **int** tipinde değer alan bir dizi oluşturduk. Çıktımızın içeriğini görmek için ekrana bastırdık. Dilimleme işlemi olarak yorum yaptığım satırda ise **a** dizisinde 2 ve 4 indeksi arasındaki değerleri dizi olarak **b**'ye kaydettik. **b** dizisinin içeriğini ekrana bastırdığımızda ise dilimlenmiş alanımızı gördük. Dilimleme işleminde **[]** içerisine dilimlemenin başlayacağı ve biteceği indeksi yazarız.

Dilim Varsayılanları (Sıfır Değerleri)

```
package main

import "fmt"

func main() {
    a := [6]int{2, 3, 5, 6, 7, 9}
    var b []int = a[:4] //Boş bırakılan indeks 0 varsayıldı
    fmt.Println(b) //Çıktımız: [2 3 5 6]
    var c []int = a[3:] //Boş bırakıldığı için 3. index ve sonrası alındı
    fmt.Println(c) //Çıktımız: [6 7 9]
}
```

Dilim Uzunluğu ve Kapasitesi

Bir dilimin uzunluk ve kapasite değeri vardır. Dilimin uzunluğunu `len()` fonksiyonu ile, kapasitesini ise `cap()` fonksiyonu ile hesaplarız. Örneğimize geçelim.

```
package main

import "fmt"

func main() {
    a := [6]int{2, 3, 5, 6, 7, 9}
    b := a[2:4]
    fmt.Println("a uzunluk", len(a))
    fmt.Println("a kapasite", cap(a))
    fmt.Println("a'nın içeriği", a)
    fmt.Println("b uzunluk", len(b))
    fmt.Println("b kapasite", cap(b))
    fmt.Println("b'nin içeriği", b)
}
```

`b` dizisi ile `a` dizisini dilimlediğimiz için `b` dizisinin kapasitesi ve uzunluğu değişti. Uzunluk dizinin içindeki değerlerin sayısıdır. Kapasite ise dizinin maksimum alabileceği değer sayısıdır.

Çıktımıza bakacak olursak;

```
a uzunluk 6
a kapasite 6
a'nın içeriği [2 3 5 6 7 9]
b uzunluk 2
b kapasite 4
b'nin içeriği [5 6]
```

Boş Dilimler (Nil Slices)

Boş bir dilimin varsayılan (sıfır) değeri **nil**'dir. Örnek olarak;

```
package main

import "fmt"

func main() {
    var a []int

    if a == nil {
        fmt.Println("Boş")
    }
}
```

Çıktısı tahmin edeceğiniz üzere **Boş** yazısı olacaktır.

Make ile Dilim Oluşturma

Dilimler **make** fonksiyonu ile de oluşturulabilir. Dinamik büyüklükte diziler oluşturabiliriz.

```
a := make([]int, 5)
```

Burada **make** fonksiyonu ile uzunluğu **5** olan **a** adında bir dizi oluşturduk.

```
a := make([]int, 0, 5)
```

Burada ise **make** fonksiyonu ile uzunluğu **0**, kapasitesi ise **5** olan **a** adında bir dizi oluşturduk.

Dilime Ekleme Yapma

Bir dilime ekleme yapmak için **append** fonksiyonu kullanılır. Hemen bir örnek ile kullanılışını görelim.

```
package main

import "fmt"

func main() {
    var a []string
    fmt.Println(a)           //[ ]
    a = append(a, "Ali")
    a = append(a, "Veli")
    fmt.Println(a)           //[Ali Veli]
}
```

a isminde **string** tipinde boş bir dizi oluşturduk. Hemen ardından boş olduğunu teyit etmek için **a** dizisini ekrana bastırdık. Daha sonra **a** dizisine **append** fonksiyonu ile **"Ali"** değerini ekledik. Yine aynı yöntem ile **"Veli"** değerini de ekledik. Son olarak **a** dizisinin çıktısının ekrana bastırdığımızda değerlerin eklenmiş olduğunu gördük.

```
fmt.Println(len(a), cap(a))
```

a dizisinin uzunluk ve capacitiesine baktığımızda aşağıdaki çıktıyı alırız.

```
2 2
```

Range

Range, üzerinde kullanıldığı diziyi **for** döngüsü ile tekrarlayabilir. Bir dilim **range** edildiğinde, tekrarlama başına iki değer döndürür (return). Birinci değer dizinin **indeksi**, ikinci değer ise bu indeksin içindeki **değer**dir. Örneğimize geçelim.

```
package main

import "fmt"

var isimler = []string{"Ali", "Veli", "Hasan", "Ahmet", "Mehmet"}

func main() {
    for a, b := range isimler {
        fmt.Printf("%d. indeks = %s\n", a, b)
    }
}
```

Yukarıdaki yazdığımız kodları açıklayalım. **isimler** isminde içerisinde **string** tipinde değerler olan bir dizi oluşturduk.

For döngümüz ile dizinimizdeki değerleri sıralayacak bir sistem oluşturduk. Döngümüzü açıklayacak olursak, bahsettiğimiz gibi dizi üzerinde uygulanan **range** terimi iki değer döndürecek olduğundan bu değerleri kullanabilmek için **a** ve **b** adında argüman belirledik. **range isimler** diyerek **isimler** dizisini kullanacağımızı belirttik. Ekranı bastırma bölümümüzde ise **%** işaretleri ile sağ taraftan hangi değerleri nerede kullanacağımızı belirttik.

Çıktımız ise şu şekilde olacaktır.

```
0. indeks = Ali
1. indeks = Veli
2. indeks = Hasan
3. indeks = Ahmet
4. indeks = Mehmet
```

MAP

Map'in Türkçe karşılığında yapacağı işlemi anlatan bir çeviri olmadığı için anlamı yerine yaptığı işi bilelim. Map ile bir değişken içerisindeki dizileri bölge olarak ayırabiliriz. Çok karmaşık bir cümle oldu. O yüzden örneğimize geçelim ki anlaşılır olsun.

```
package main

import "fmt"

type insan struct {
    kisi1, kisi2, kisi3 string
}

func main() {
    var m map[string]insan
    m = make(map[string]insan)
    m["isim"] = insan{
        "Ali", "Veli", "Ahmet",
    }
    fmt.Println(m["isim"])
}
```

Yukarıda **insan** isminde bir struct metodu oluşturduk ve içerisine **string** tipinde 3 tane değişken girdik. main() fonksiyonumuz içerisinde ise **m** adında **map** kullanarak **string** değer saklayabilen **insan** tipinde değişken oluşturduk. **m** değişkenini **make** ile **map** dizisi haline getirdik. Hemen aşağısında ise **m** değişkenine **"isim"** adında bir bölge oluşturduk ve **insan** struct'ında belirttiğimiz gibi 3 tane string değer girdik. Son olarak **m** dizisinin isim bölgesindeki değerleri ekrana bastırmasını istedik. **Çıktımız şöyle olacaktır;**

```
{Ali Veli Ahmet}
```

Birden Fazla Bölge Ekleme

Önceki yazımızda **map** ile dizilere bölgesel hale getirmeyi gördük. Şimdi de birden fazla bölgeyi nasıl yapacağımızı göreceğiz. Örneğimize geçelim.

```
package main

import "fmt"

type insan struct {
    kisi1, kisi2, kisi3 string
}

var m = map[string]insan{
    "erkekler": insan{"Ali", "Veli", "Ahmet"},
    "kadinlar": insan{"Ayşe", "Fatma", "Hayriye"},
}

func main() {

    fmt.Println(m["erkekler"])
    fmt.Println(m["kadinlar"])
    fmt.Println(m)
}
```

Yukarıda önceki örneğimizdeki gibi **insan** struct'ı oluşturduk ve içine 3 tane **string** tipinde değer atadık. **m** adında dizi oluşturduk ve **map** ile bölgesel bir dizi olduğunu belirttik. Dizinin içerisine “**erkekler**” isminde **insan** tipinde bir bölge oluşturduk ve içine 3 tane **string** tipinde değerimizi girdik. Aynı işlemi “**kadinlar**” isimli bölge içinde yaptık. **main** fonksiyonumuz içerisinde **erkekler** ve **kadinlar** bölgemizi ekrana bastırdık. Son olarak **m** dizisindeki tüm içeriği ekrana bastırık.

Çıktımız ise şöyle olacaktır;


```
{Ali Veli Ahmet}  
{Ayşe Fatma Hayriye}  
map[erkekler:{Ali Veli Ahmet} kadınlar:{Ayşe Fatma Hayriye}]
```

Burada ayrıntıyı farkedelim. **m** dizisini ekrana bastırdığımızda map yeni bölgeyi bir dizi olduğunu vurguluyor. Map ile bir bakıma dizi içerisine yeni bir dizi ekliyorsunuz. Tabi bunu struct metodu ile yapıyoruz.

Bölgesel Silme İşlemi

delete fonksiyonu ile silme işlemimizi yapabiliriz. Hemen örneğimize geçelim.

```
package main  
  
import "fmt"  
  
func main() {  
    m := make(map[string]int) //m isminde string bölge isimli int değer taşıyan dizi  
  
    m["sayi"] = 25 //sayi bölgesine 25 değerini yerleştirdik  
    fmt.Println(m["sayi"]) //Çıktımız: 25  
    delete(m, "sayi") //sayi bölgesindeki değeri sildik  
    fmt.Println(m["sayi"]) //Çıktımız: 0 (sıfır)  
}
```

Arayüz

Arayüz (interface) sayesinde fonksiyonlardan dönen değerin tipini başka bir yerde kullanmak için şekillendirebilir. Arayüzler neslere arasındaki iletişimi sağlamak için kullanılır. Daha anlaşılır olması için örnek üzerinden inceleyelim.

```
package main

import "fmt"

type iletisim interface {
    topla() int
}

type sayilar struct {
    bir, iki int
}

func main() {
    var a iletisim
    v := sayilar{3, 4}

    a = &v
    fmt.Println(a.topla())
}

type sayi int

func (f sayi) topla() int {
    return int(f)
}

func (sayi *sayilar) topla() int {
    return sayi.bir + sayi.iki
}
```

Yazdığımız kodları inceleyelim. **iletisim** isminde **topla()** fonksiyonundan **int** değer döndüren bir **interface** (arayüz) oluşturduk. **sayilar** adında içinde **bir** ve **iki** isminde **int** tipinde

değişken barındıran bir **struct** oluşturduk. **main** fonksiyonumuzda **a** isminde **iletisim** tipinde bir değişken oluşturduk. Daha sonra **v** isminde **sayilar** struct'ı için içerisinde **bir** ve **iki** değişkenine denk gelen **int** tipinde sayılarımızı girdik. İşaretçiler konumuzda gördüğümüz yöntem ile **a** değişkenini **pointer** yöntemi ile **v** değişkenine işaretledik. Ekranı bastırma kısmını açıklamak için aşağıdaki fonksiyonlara değinelim. **sayi** isminde bir **int** tür oluşturduk. Altındaki **topla()** fonksiyonu ile girilen sayıyı **int** tipine çeviren bir fonksiyon yazmış olduk. **interface** kullanmamızın ana mantığı da burada ortaya çıkıyor. **struct** ile bir tür oluşturduğumuzda içine girilen değerler **int** tipinde olsa bile dışarı çıkarılırken dizi şeklinde çıktığı için interface ile fonksiyonun dizi şeklinde değilde **int** tipinde veriyi kullanmasını sağlıyoruz. En son fonksiyonumuzda ise **sayilar** struct'ımızı fonksiyona dahil edip struct'ı **sayi** argümanı ile kullanacağımızı belirttik. Böylelikle **bir** ve **iki** değişkenine denk gelen sayıları toplayıp return edebildik. **main** fonksiyonunun son bölümünde ise ekrana toplamı bastırmış olduk. Çıktımız ise **7** olacaktır.



BÖLÜM 4

- Goroutine
- Kanallar
- Print Fonksiyonu
- Formatlar ve Kaçış Karakterleri
- Scan Fonksiyonu
- Variadic Fonksiyonlar
- Closure (Anonim Fonksiyonlar)
- Recursive (İç-içe) Fonksiyonlar

Goroutine

Goroutine'ler **Go Runtime** tarafından yönetilen hafif bir sistemdir. Bir işlemi **Goroutine** ile gerçekleştirmek istiyorsak o satırın başına **go** yazmamız yeterlidir. Şaşırtıcı ama sadece **go** yazarak bu işlemi yapabiliyoruz. Aynı defer'deki gibi. **Goroutine** aslında bir fonksiyon gibi çalışır. Eş zamanlı çalışacak fonksiyonları çağırmak için kullanılır. Basit bir örnek ile anlaşılır bir sonuç elde edelim.

```
package main

import (
    "fmt"
    "time"
)

func yaz(s string) {
    for i := 0; i < 5; i++ {
        time.Sleep(1000 * time.Millisecond)
        fmt.Println(s)
    }
}

func main() {
    go yaz("Dünya")
    yaz("Merhaba")
}
```

Yazdığımız kodları inceleyelim. Zaman ile alakalı fonksiyonları kullanabilmek için **time** paketini import ettik. **yaz** fonksiyonu oluşturduk. Bu fonksiyon **s** isminde **string** tipinde değeri işleyecek. Fonksiyonun bloğunda **5** defa **1** saniye bekleyerek istenilen yazıyı ekrana bastıran kodlarımızı girdik. **main()** fonksiyonumuzda bir tane iki tane **yaz()** fonksiyonu kullandık. Birinin başına **go** terimini ekledik.

Çıktımız şu şekilde olacaktır;

Merhaba
Dünya
Merhaba
Dünya
Dünya
Merhaba
Merhaba
Dünya
Dünya
Merhaba

Çıktımız ekrana bastırılırken belirlediğimiz zaman beklemesi ile birlikte çıkar. **go** terimini kaldırıp veya erteleme zamanını değiştirerek deneyde bulunabilirsiniz.

Kanallar

Kanallar, nesneler arasında `<-` işareti ile veri alış-verişi yapabildiğimiz hatlardır. Kanallarında diziler gibi `make()` ile oluşturabiliriz. `make()` fonksiyonunun dinamik kapasite oluşturması sayesinde işimiz kolaylaşır. Kanallar kullanılmadan önce oluşturulmalıdır.

```
package main

import "fmt"

func topla(s []int, c chan int) {
    toplam := 0
    for _, v := range s {
        toplam += v
    }
    c <- toplam //toplam değerini c kanalına yolladık
}

func main() {
    s := []int{7, 2, 8, -9, 4, 0}

    c := make(chan int)
    go topla(s[:len(s)/2], c)
    go topla(s[len(s)/2:], c)
    x, y := <-c, <-c //c kanalından veri aldık

    fmt.Println(x, y, x+y)
}
```

`topla` adında bir fonksiyon oluşturduk. `s` adında dizi değişkeni ve `c` adında `int` tipinde bir kanal kullanacağımızı belirttik. Fonksiyonumuz içerisinde `toplam` adında `0` değerinde sayısal bir değişken oluşturduk. `Range`'i anlattığım dersten hatırlayacağınız üzere, `range`, `for` döngüsü ile bir dizi üzerine uygulandığında 2 farklı değer döndürüyordu. Burada bize `indeks` lazım olmadığından

`indeks` yerine belirleyeceğimiz değer yerine `_` (alt tire) kullanarak kullanıma kapattık. Gelen değerın dizi uzunluğu ile aynı kere toplam değişkenine eklenmesini sağladık. `for` döngümüz bittikten sonra `toplam` içindeki değeri `c` kanalına yolladık.

`main()` fonksiyonumuz içerisinde `s` adında içerisinde `int` tipinde değerler barındıran bir dizi oluşturduk. `make` fonksiyonunu kullanarak `c` adında `int` tipinde bir kanal oluşturduk. `make` ile oluşturduk ki dinamik boyutta olabilsin. `go` ile farklı dilimlemeler ile `topla` fonksiyonlarını çalıştırdık. Böylece `goroutine`'e birden fazla thread açtık. `c` kanalından verilerimizi alırken `x` ve `y` değişkenleri için farklı değer almış olduk. En son tüm çıktımızı ekrana bastırdık ve çıktımız bu şekilde oldu;

```
-5 17 12
```


Print Fonksiyonu

Print fonksiyonu Go dilinde komut satırı üzerinde yazdırma işlemi yapmak için kullanılır. Print fonksiyonunun en çok kullanılan 3 çeşidine bakalım.

Print() Fonksiyonu

Bu fonksiyonun içine parametreler girerek ekrana yazdırma işlemi yapabiliriz.

```
fmt.Print("Merhaba Dünya!")
```

Çıktımız şu şekilde olacaktır;

```
Merhaba Dünya!
```

Println() Fonksiyonu

Bu fonksiyon ile içine parametre girerek ekrana yazdırma işlemi yapabiliriz. Yazdırma işlemi yaptıktan sonra bir alt satıra geçer.

```
fmt.Println("satır1")  
fmt.Println("satır2")
```

Çıktımız şu şekilde olacaktır;

```
satır1  
satır2
```

Printf() Fonksiyonu

Gelelim işimizi göreceğ olan Printf() fonksiyonuna. Bu fonksiyon sayesinde metinsel bölümlerin arasına değişken yerleştirebiliriz.

```
dil:="Go"  
yıl:=2007  
  
fmt.Printf("%s dili %d yılından beri geliştiriliyor.",dil,yıl)
```

Çıktımız şu şekilde olacaktır;

Go dili 2007 yılından beri geliştiriliyor.

Yazdığımız fonksiyonu inceleyelim. **dil** adında bir değişken belirledik ve değerini “Go” verdik. **%** işaretinin yanına kullanacağımız değişkenin türüne göre formatımızı belirttik.

Formatların Kullanım Alanları

Format	Açıklama
%T	Değişkenin tipini belirtir.
%t	Boolean değeri belirtir.
%d	Int değeri belirtir.
%b	Sayının binary karşılığını belirtir.
%c	Karakter değerini belirtir.
%x	Sayının hexadecimal karşılığını belirtir.
%f	Float değeri belirtir.
%s	String değeri belirtir.

Kaçış Karakterleri

Kaçış karakterleri string metinlere etki eden işaretlerdir. Kaçış karakterleri tablosunu görelim.

Kaçış Karakteri	Açıklama
\a	Komut satırında zil sesi çıkarır.
\b	Silme tuşu görevini görür.
\f	Merdiven metin yazar.
\n	Satır atlatır.
\r	Return eder.
\t	Tab tuşu gibi boşluk bırakır.

Kaçış Karakteri	Açıklama
\v	Dikey sekme boşluğu bırakır.
\\	Ters-taksim yapar.
\'	Alıntı işareti yapar.
\"	Çift tırnak işareti yapar.

Birkaç örnek yapalım da, anlamlı olsun.

```
fmt.Println("\aZil Çalma")  
fmt.Println("\tSekme Bırakma\nSatır Atlama")  
fmt.Println("Merdiven\fYazı\fYazma")
```

Çıktımız şu şekilde olacaktır;

```
Zil Çalma(Zil Çalma Sesi Gelir)  
    Sekme Bırakma  
Satır Atlama  
Merdiven  
    Yazı  
        Yazma
```

Scan Fonksiyonu ile Kullanıcıdan Giriş Alma

Scan() fonksiyonu ile komut satırı üzerinden kullanıcı programımıza girişte bulunabilir. Böylece programımızı interaktif hale getirmiş oluruz. 3 tane Scan türünü inceleyelim.

Scan() Fonksiyonu

Bu fonksiyon ile kullanıcı giriş yerine **sadece bir kelime** girebilir. Örnek kullanımı aşağıdaki şekildedir.

```
var yazi string

fmt.Scan(&yazi)
fmt.Println("\n"+yazi)
```

Yukarıda yazdığımız kodları inceleyecek olursak, belleğe **yazi** isimli **string** türünde bir değişken kaydettik. Kullanıcının girişte bulunabilmesi için **Scan()** fonksiyonunu kullandık. Bu fonksiyonun içerisine **&yazi** yazdık. Bu sayede kullanıcının girdiği değer **yazi** değişkeninin içerisine kaydedilebilecek. Daha sonra **yazi** değişkenini ekrana bastırdık ve bizim yazdığımız değer görüntülendi. **Scan** fonksiyonunda dikkat edilmesi gereken nokta kullanıcı istediği kadar kelime girse bile programın ilk kelimeyi değer olarak alacağıdır. Scan() fonksiyonu boş giriş kabul etmez.

Scanln() Fonksiyonu

Scanln() fonksiyonun farkı kullanıcının kelime sınırı olmamasıdır. Örneğimizi görelim.

```
var yazi string

fmt.Scanln(&yazi)
fmt.Println("\n"+yazi)
```

Kullanıcının girdiği herşey **yazi** değişkeni içerisine kaydedilecektir. Kelime sınırı yoktur. Boş giriş kabul edilmez.

Scanf() Fonksiyonu

Scanf() fonksiyonu **Printf()** fonksiyonu gibi **format** içerir. Bu fonksiyon ile kullanıcının girişini bölüp birkaç değişkene kaydedebiliriz. Hemen kullanımını görelim.

```
var kelime1, kelime2 string

fmt.Scanf("%s %s",&kelime1,&kelime2)
fmt.Println(kelime1)
fmt.Println(kelime2)
```

Yukarıda yazdığımız kodları inceleyecek olursak, **kelime1** ve **kelime2** adında **string** türünde değişkenler belirledik. **Scanf()** fonksiyonu ile **Printf()**'den benzer olarak, değişkenlerin yerleştirileceği yerleri değil de, bu sefer değişkenlerin alınacağı yerleri belirtiyoruz. **%s %s** arasındaki boşluk sayesinde kullanıcı boşluk bırakınca girdiyi 2 değere bölebileceğiz. Hemen yanında ise içine atanacak değişkenlerimizi belirtiyoruz. Böylelikle kullanıcı giriş bölümünden **Go Dili** yazdığında **Go**'yu **kelime1**'in içine **Dili** de **kelime2** içine yerleştirecek. **Scanf()**, boş giriş kabul eder.

Scan Fonksiyonu Birkaç Tüyo

Eğer **build** ettiğimiz programın işlemler bittikten sonra kapanmamasını istiyorsanız, Kodların sonuna **Scan** fonksiyonu ekleyebilirsiniz. Böylelikle **[ENTER]** tuşuna basınca programımız kapanır. **Scan()** ve **Scanln()** fonksiyonu boş giriş kabul etmediği için bu işlemi **Scanf()** fonksiyonu ile yapabiliriz. Çünkü **Scanf()** içerisine birşey yazılmadan **[ENTER]** tuşuna basıldığından boş değeri kabul eder. Fonksiyondaki girişi hiçbir değişkene atamadan şöyle yazabiliriz.

```
fmt.Scanf("%s")
```

Variadic Fonksiyonlar

Variadic fonksiyon tipi ile fonksiyonumuza kaç tane değer girişi olduğunu belirtmeden istediğiniz kadar değer girebilirsiniz. Hemen örneğimize geçelim.

```
package main

import "fmt"

func main() {
    fmt.Println(toplama(3, 4, 5, 6)) //18
}

func toplama(sayilar ...int) int {
    toplam := 0
    for _, n := range sayilar {
        toplam += n
    }
    return toplam
}
```

Yukarıdaki fonksiyonumuzu inceleyelim. Vericeğimiz sayıları toplamaları için aşağıda **toplama** adında bir fonksiyon oluşturduk. Fonksiyonun parametresi içerisine, yani parantezler içerisine, **sayilar** isminde **int** tipinde bir değişken tanımladık. **...** (üç nokta) ile istediğimiz kadar değer alabileceğini belirttik. **toplam** değerini mantıken doğru değer vermesi için **0** yaptık. Çünkü her sayıyı toplam değişkeninin üzerine ekleyecek. **range**'in buradaki kullanım amacından bahsedeyim. **range**'i **for** döngüsü ile kullandığımızda işlem yaptığımız öğenin uzunluğuna göre işlemimizi sürdürürüz. Yani fonksiyonumuzun içine ne kadar sayı eklersek işlemimiz ona göre şekillenecektir. **Range** kullanımında **_**, **n** şeklinde değişken tanımlamamızın sebebi, birinci değişken yani **_**, dizinin indeksini yani sıra numarasını verir. Bizim bununla bir işimiz olmadığı için **_** koyarak kullanmayacağımızı belirttik. İkinci değişken ise yani **n** dizinin içindeki değeri verir yani fonksiyona girdiğimiz sayıları. Sonuç olarak bu fonksiyonda **return** ile **for** işleminden sonra tüm sayıların toplamını döndürüp **main()** fonksiyonu içerisinde ekrana bastırmış olduk.

Closure (Anonim) Fonksiyonlar

Closure fonksiyonlar ile değişkenlerimizi fonksiyon olarak tanımlayabiliriz. Örneğimize geçelim.

```
package main

import "fmt"

func main() {
    toplam := func(x, y int) int {
        return x + y
    }
    fmt.Println(toplam(2, 3))
}
```

Yukarıdaki kodlarımızı inceleyecek olursak, **main** fonksiyonunun içine **toplam** adında bir değişken oluşturduk. Bu değişkenin türünün otomatik algılanması için **:=** işaretlerimizi girdik. Değişkene değer olarak anonim bir fonksiyon (ismi olmayan fonksiyon yani) yazdık. Bu fonksiyon **x** ve **y** adında iki tane **int** değer alıyor ve return kısmında bu iki değeri int olarak döndürüyor. Aşağıdaki **Println()** fonksiyonunda ise bu değişkeni aynı bir fonksiyonmuşcasına kullandık.

Recursive (İç-içe) Fonksiyonlar

Recursive fonksiyonlar yazdığımız fonksiyonun içinde aynı fonksiyonu kullanmamız demektir. Fonksiyonumun tüm işlemler bittiğinde return olur. Örneğimize geçelim.

```
package main

import "fmt"

func main() {
    fmt.Println(faktoriyel(4))
}

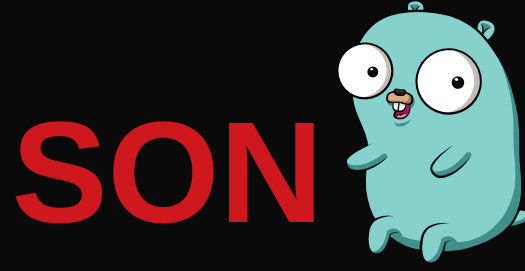
func faktoriyel(a uint) uint {
    if a == 0 {
        return 1
    }
    return a * faktoriyel(a-1)
}
```

Yukarıdaki fonksiyon ile bir sayının faktöriyelini hesaplayabiliriz. Faktöriyel hakkında kısaca bir hatırlatma yapayım. Belirlediğimiz sayıya kadar olan tüm sayıların sırasıyla çarpımına o sayının faktöriyeli denir. Yani 4 sayısının faktöriyelini bulmak istiyorsak: $1*2*3*4$ işlemini yaparız. Sonuç 24'tür. Faktöriyel fonksiyonun giriş ve çıkış tiplerini **uint** yapmamızın sebebi ise faktöriyel sonucunu bulmak için en geriye gidildiğinde eksi değerlere geçilmemesi içindir. Ayrıca sıfırın faktöriyeli birdir. Onun için değer sıfırsa bir return etmesini istedik. Faktöriyel fonksiyonunun en alttaki **return** kısmında girdiğimiz sayı ile girdiğimiz sayının bir eksiğinin faktöriyelini çarpacak. Girdiğimiz sayının bir küçüğünü bulmak içinse yeniden o sayının faktöriyelini hesaplayacak. Daha sonra aynı işlemler bu sayılar içinde yapılacak. Taki sayı son geldiğinde yani en küçük **uint** değeri olan 0'a dayandığında. Daha sonra sonucu **main** fonksiyonu içerisinde ekrana bastırdık.



KAYNAKÇA

- **GODOC**
- **Golang Tour**



Lütfen Okuyunuz!

Umarım Golang'ın temelini düzgün bir şekilde anlatabilmişimdir. Yazdığım kaynak hakkında görüşlerinizi, hataları ve önerilerinizi bildirmek için GitHub adresini aşağıya bıraktım. GitHub üzerinden kaynağın düzenlenmiş halini güncel tutmaya çalışacağım. GitHub deposuna Golang ile alakalı örnekler koymaya çalışacağım. Son versiyonu için GitHub'ı kontrol etmeyi unutmayın. Her zaman eleştiriye açığım. Bilginize..

GitHub: <https://github.com/ksckaan/golangtrpdf>

İletişim: kaanksc@hotmail.com

ksc10