DART DİLİ

Blok yapısı:

```
void main() {


 }
```

Veri Tipleri : Tüm veri tipleri Object yani nesnedir.

Veri tiplerinde Null değeri kullanılması istenmez. İllaki kullanmak gerekirse;

```
int? i = null; // null eklemek gereksizdir.
int? j;  // Dart'ta, açıkça başlatılmayan bir değişken otomatik olarak null olarak başlatılır.
print(i);
print(j);
```

var: Veri tipi belirtilmek istenmediği zaman "var" (varaible) kullanılır. Veri tipi atanan ifadeye göre otomatik belirlenir.

```
var d1=5;
var d2=5.6;
var d3="Mobil";
var d4=true;
```

Veri tipini kendimiz belirtmek istersek;

num: Number veri tipi hem tamsayı hemde ondalıklı sayıları tanımlamak için kullanılabilir.

int: Tamsayılar

double: Ondalıkla sayılar

String: Karakter ve metinsel ifadeler

bool: Boolean, Doğru veya Yanlış

Object veya dynamic : Tek bir tiple sınırlı olmayan değerler

```
num d1=5;   num d2=5.6;  d1=3.5;  d2=5;
int d3=5; d3=5.2
double d4=5.6; d4=6;
String d5="Mobil";
bool d6=true;
Object d7="Mobil";  d7=5;  d7=2.8;
dynamic d8="Mobil";  d8=false;
var d9=5; d9=3.4;
```

Değişkenlerin Metodları:

Bir değişkenin isminin yanına nokta konulduğu zaman ona ait metodlar karşımıza gelir ve istenilen kullanılabilir.

```
int sayi=22;

print(sayi.isEven);   // Sayının çift olup olmadığına bakar. Burada True sonucu döner.
```

print() Komutu: Ekrana yazdırma

```
print("Mobil");
print('Mobil');
int a=5;
print(a);
print(a+12);
String ders="Mobil";
print(ders);
print(ders[3]);   //  i
print(ders + "Programlama");     // + işareti değerlerin birleşik yazılmasını sağlar
print(ders + ' ' + "Programlama");
print('Mobil' + "Programlama");
print('Mobil' + ' '+ "Programlama");
print('Mobil\' Programlama');
print("Mobil' Programlama");
var i="Mobil' Programlama";   //    var i='Mobil\' Programlama';
print(i);
```

Interpolation Durumu: metin biçimlendirmek için kullanılmaktadır.

```
String ders1="Mobil";

String ders2="Programlama";

print("$ders1 $ders2");   //  $ işareti sayesinde "  " işaretleri arasında değişkenler kullanılabilir.

print("$ders1 kelimesi" + ' ' + ders1.length.toString()+ ' karakterdir');     // veya

print("$ders1 kelimesi ${ders1.length} karakterdir");  // { } işareti ile "  " işaretleri arasında değişkenlerin ek özelliklerini kullanılabilir.

print("${(ders1+' '+ders2).length} karakterdir");   //   () işaretleri { } işareti arasında birden fazla değişkenin ek özellikleriyle işlem yapmak için kullanılabilir.
```

```
double a=5.5;
double b=10;
print("$a ve $b'nin toplamı = ${a+b}");
print("$a ve $b'nin toplamı = ${(a+b).toInt()}");
```

List (Listeler): Sıralı Veri Koleksiyonu

```
List isim = [];  // Boş bir liste oluşturur.   List<dynamic>   dynamic: her tipten eleman olabilir anlamında.
List isim1 = ["ali","mehmet"];
print(isim);
print(isim1);   //   [ali, mehmet]
List isim2 = ["ali","mehmet",12,5.6,true];
print(isim2);   //   [ali, Mehmet, 12, 5.6, true]


List<String> a = ['5'];   // Eğer listeyi belli bir veri tipinde oluşturmak istersek belirtmeliyiz.
print(a);
```

Listeler kendi içerisinde boş veya dolu başka listeler alabilir.

```
List a = ['ali',5,[],3,[4.2,'mehmet']];
print(a);   //  [ali, 5, [], 3, [4.2, mehmet]]
print(a[3]);   //  3
print(a[2]);   //  []
print(a[4]);   //  [4.2, mehmet]
print(a[4][1]);   //   mehmet
print(a[4][1][2]);   //    h
a[1]=15;    //    [ali, 15, [], 3, [4.2, mehmet]]
print(a);
a.add(8);   // liste sonuna ekler  [ali, 15, [], 3, [4.2, mehmet],8]
print(a);
a.removeAt(1);  //  belirlenen pozisyondakini siler  [ali, [], 3, [4.2, mehmet],8]
print(a);
a.remove('ali');   //  verilen elemanı siler   [[], 3, [4.2, mehmet],8]
print(a);
a.clear();   //  listeyi tümden siler   []
print(a);
```

Set (Kümeler): Sırasız Veri Koleksiyonu

Kümelerde listelerden farklı olarak aynı elemandan iki tane bulunamaz.

```
Set numaralar = {1,2,3};
print(numaralar);  // {1, 2, 3}
print(numaralar.contains(2));  // true     değerin var olup olmadığı kontrol edilir
print(numaralar.contains('2'));  // false


Set kisiListesi = {'Mehmet', 'Ahmet', 'Burak', 'Ayşe'};
Set gelenKisiler = {'Mehmet', 'Şafak'};
print(kisiListesi.intersection(gelenKisiler));  //  {Mehmet}    Ortak olanlar (Kesişim) verilir
print(kisiListesi.union(gelenKisiler));  //  {Mehmet, Ahmet, Burak, Ayşe, Şafak}    Birleşim kümesi verilir
```

Map (Sözlükler): Anahtarlı Veri Koleksiyonu

Anahtar: Değer şeklinde ifade edilir.

```
Map sozluk={'Ali':'1213001','Mehmet':'1213005'};    //  Map<dynamic, dynamic> sozluk
print(sozluk);    //    {Ali: 1213001, Mehmet: 1213005}
Map sozluk2={'Ali':90,'Mehmet':'1213005'};
print(sozluk2);  //   {Ali: 90, Mehmet: 1213005}
```

Sözlüklerde de hem anahtar hem değer için veri tipi tanımlaması yapabiliriz.

```
Map<String, String> sozluk={'Ali':'1213001','Mehmet':'1213005'};
Map<String, int> sozluk2={'Ali':90,'Mehmet':1213005};
print(sozluk['Ali']);   //    1213001  Sözlük içindeki değerler ulaşmak için anahtar kelime kullanılır.
sozluk['Hasan']='121385';   //    Sözlük sonuna yeni bir eleman ekler
print(sozluk);    //    {Ali: 1213001, Mehmet: 1213005, Hasan: 121385}
sozluk.remove('Mehmet');    //  verilen anahtar değer silinir
print(sozluk);     //     {Ali: 1213001, Hasan: 121385}
```

Sözlük içinde liste ve sözlük kullanabiliriz.

```
Map sozluk={'a':'Ali','b':90,'c':[1,4,'xyz'],5:{'ad':'ali'} };
print(sozluk);    //    {a: Ali, b: 90, c: [1, 4, xyz], 5: {ad: ali}}
print(sozluk.keys);  // anahtarlar listelenir
print(sozluk.values);  // değerler listelenir
```

if () : Şart cümleleri

```
int sayi1 = 5, sayi2 = 5;
if (sayi1 > sayi2) {
  print("$sayi1 > $sayi2 dir");
} else if (sayi2 > sayi1) {
  print("$sayi2 > $sayi1 dir");
} else {
  print("$sayi1 = $sayi2 dir");
}
```

Kısa if kullanımı:

```
int sayi1 = 7, sayi2 = 5, kucuk;
sayi1 < sayi2 ? kucuk = sayi1 : kucuk = sayi2;
print (kucuk);


kucuk = sayi1 < sayi2 ? sayi1 : sayi2;
print (kucuk);


int? sayi3, sayi4 = 5, deger;
deger = sayi3 ?? sayi4 ;     //  ?? işaretleri değişkenlerin sırasıyla null durumuna bakar. Null olmayan değişken değeri, deger değişkenine aktarılır.
```

switch() : Şart cümleleri

```
int a=2;
switch(a){
  case 1:
    print ("Bir");
    break;
  case 2:{
    print ("İki");
    break;
  }
  default:
    print("Hatalı değer");
}
```

for() : Döngü

```
for (int i = 0; i < 5; i++) {
  print('Mobil ${i + 1}');
}

List harf=["A","B","C"];
for (String x in harf){
  print(x);
}
print(harf.length);
for (int i=0;i<harf.length;i++){
  print(harf[i]);
}
```

while() : Döngü

```
int i=0;
while (i<3){
  print('Mobil ${i + 1}');
  i++;
}

i=0;
do {
  print('Mobil ${i + 1}');
  i++;
}while (i<3);
```

break : Döngü sonlanır.

continue : Döngü başına gider.

Fonksiyonlar:

Aşağıda, ornek adında bir fonksiyon oluşturulması ve main içerisinde kullanımı 2 farklı şekilde gösterilmektedir.

1-

```
void main() {
 ornek() {
   print('Mobil');
 }

 ornek();
}
```

2-

```
void main() {
  ornek();
}

ornek() {
  print('Mobil');
}
```

Fonksiyonlarda parametre kullanımı; ZORUNLU PARAMETRE

```
void main() {
  ornek('Mobil');
}

ornek(parametre) {
  print(parametre);
}
```

```
ornek(int parametre) {
  print(parametre);
}
```

Parametre değişkeninin veri tipini belirtirsek, fonksiyona ancak o veri tipinde parametre gönderebiliriz.

Farklı bir örnek;

```
void main() {
  topla(3,5);
}

topla(int x, int y) {
  print(x+y);
}
```

return ile değer döndürmek;

```
void main() {
  print(topla(3,5));
}

topla(int x, int y) {
  return x+y;
}
```

```
int topla(int x, int y) {
  return x+y;
}
```

return ile değer döndüren bir fonksiyonun, geriye hangi tipte değer döndüreceği belirtilebilir.

Değer döndürmeyen fonksiyonlar void tipindedir.

Aynı işlemi farklı olarak ifade edecek olursak;

```
void main() {
  var sonuc = topla(3, 5);
  print(sonuc);
}

int topla(int x, int y) {
  return x + y;
}
```

Fonksiyonlarda parametre kullanımı; KEYFİ (MECBURİ OLMAYAN) PARAMETRE

```
void main() {
  goster(5,3);          //  X = 5, Y = 3
}

goster(int x, int y) {
  print('X = $x, Y = $y');
}
```

```
void main() {
  goster(5);          //  X = 5, Y = null
}
goster(int x, [int? y]) {
  print('X = $x, Y = $y');
}
```

Keyfi parametreler [ ] içerisine alınır.

Parametrelere varsayılan değer verebiliriz;

```
void main() {
  goster(5);              //   X = 5, Y = 8

}

goster(int x, [int y=8]) {
  print('X = $x, Y = $y');
}
```

Parametrelerin gönderim sırası; (isimli parametre)

Normalde parametreler hangi sırayla oluşturulduysa o sırayla göndermek gerekir. Farklı olarak isimli parametre gönderimi yapabiliriz. Bu durumda sıraya uymak gerekmez.

İsimli parametre kullanılabilmesi için fonksiyonda parametreler tanımlanırken { } işaretleri içerisine alınmalıdır. İsimli parametrede parametreler tercihe bağlıdır.

```
void main() {
  goster(y:5);              //     X = null, Y = 5
  goster(y:5, x:3);         //     X = 3, Y = 5
  goster();                 //     X = 22, Y = 33
}

goster({int x=22, int y=33}) {
  print('X = $x, Y = $y');
}
```

Fonksiyonun "sağa ok" ile tanımlanması:

```
void main() {
  goster();             //   7
}

goster() => print(2 + 5);
/*
goster() {
  print(2 + 5);
}
*/
```

Başka bir örnek;

```
void main() {
  print(topla(2,5));
}

topla(int x, int y) => x+y;
/*
topla(int x, int y){
  return x+y;
}
*/
```

```
int topla(int x, int y) => x+y;      // 7  Sadece tamsayı döndürür
topla(int x, int y) => print('Mobil');     // komut döndürür
topla(int x, int y) => 2;       //  2
topla(int x, int y) => 'Mobil';      //  Mobil
```

Anonim Fonksiyonlar:

() {print('Mobil');} // Anonim fonksiyonlar ismi olmayan fonksiyonlardır.

Bir anonim fonksiyonu, başka bir fonksiyonda kullanmak için bir değişken ismiyle atanabilir. Örneğin;

```
void main() {
  yaz();            //  Mobil
}
var yaz=(){print('Mobil');};      // veya   var yaz=()=>print('Mobil');
```

Başka örnek;

```
void main() {
  yaz(2,6);           //  8
}
var yaz=(x,y)=>print(x+y);
```

Anonim fonksiyonun başka bir fonksiyona parametre olarak gönderilmesi;

```
void main() {
  yaz(topla);
}
var topla=(x,y) => print(x+y);
yaz(gonderilen) {gonderilen(5,8);}
```

```
void main() {
  yaz((x, y) => print(x + y));
}
yaz(gonderilen) {gonderilen(5, 8);}
veya bu şekildede kullanılabilir.
```

Başka bir örnek;

```
void main() {
  List liste=['Ali','Mehmet'];
  liste.forEach((parametre)=>print(parametre));
}
```

```
Ali
Mehmet
```

Liste veri tipinin forEach özelliği (void forEach(void Function(dynamic) action)) parametre alan ama değer döndürmeyen bir fonksiyon ister. Listedeki değerleri bu fonksiyona tek tek gönderir.

SENKRON VE ASENKRON İŞLEMLER

İşlemler sırasıyla yapılıyor ve bir işlem tamamlanmadan diğer bir işlem başlamıyorsa buna senkron denir. Bekleme yapmayan işlemlere ise asenkron denir. Yani bir işlemin tamamlanması beklenmeden yeni gelen diğer işlemlerin sıraya alınması ve işlemlerin arka planda tamamlanmasıdır. Bir işlemin tamamlanması beklenirken diğer işlemlerin devam etmesine olanak tanır.

Programlamada bizi bekleten bazı süreçler vardır. Örneğin, internetteki bir kaynaktan veri çekmek, veri tabanına veri yazmak, dosyadan veri okumak gibi.

Örnek; siparisHazirla adında ASENKRON bir fonksiyon.

```
void main() {
}
siparisHazirla(){
  return Future.delayed(Duration(seconds:4), (){return 'Siparişiniz alındı';});
  //return Future.delayed(Duration(seconds:4), ()=>'Siparişiniz alındı');
}
```

Future() fonsiyonu:

Future.delayed metodu fonksiyonun gecikme yapmasını sağlar.

(new) Future<dynamic> Future.delayed(Duration duration, [FutureOr<dynamic> Function()? computation])

Delayed metodu iki tane parametre alır. İlk parametre (Duration) bekleme süresini, ikinci parametre bekleme tamamlandığı zaman çalışacak olan anonim fonksiyonu belirler.

(new) Duration Duration({

int days = 0, int hours = 0, int minutes = 0, int seconds = 0, int milliseconds = 0, int microseconds = 0, })`

Duration, gün, saat, dakika, saniye, milisaniye ve mikrosaniye parametreleri alır. Bu parametreler isimli parametre. Bu nedenle parametrelerin isimli olarak gönderilmesi gerekir.

SiparişHazırla fonksiyonu çalıştırıldığında 4 saniye bekler ve sonra ‘Sipariş Alındı’ değerini döner.

Şimdi bir tanede ayrı bir siparisVer fonksiyonu yazalım ve bu foksiyonda siparisHazırla fonksiyonundan dönen değeri bir değişkene alıp ekrana yazdıralım. Bunu mainde çalıştıralım.

```
void main() {
  siparisVer();
}
siparisVer(){
  var siparisDurumu=siparisHazirla();
  print('Sipariş Durumu = $siparisDurumu');
}
siparisHazirla(){
  return Future.delayed(Duration(seconds:4), (){return 'Siparişiniz alındı';});
  //return Future.delayed(Duration(seconds:4), ()=>'Siparişiniz alındı');
}
```

Çalıştırıldığı zaman ekranda : `Sipariş Durumu = Instance of '_Future<String>'` sonucu alınır. Bize bir Future örneği döndürür. Tamamlanmamış işlemi temsil eder.

Bunun nedeni: siparişVer fonksiyonu çalıştırıldığında siparisHazırla fonksiyonundan dönecek olan değer siparisDurumu değişkenine aktarılamaz çünkü siparisHazirla fonksiyonu değeri 4 saniye sonra döndürecektir. Ama

siparisVer fonksiyonu, siparisHazirla fonksiyonunun tamamlanmasını beklemeden bir sonraki satıra geçer. Bu nedenle siparisHazirla fonksiyonu vermesi gereken değeri veremez.

Burada siparişVer fonksiyonu, siparisHazirla fonksiyonunun tamamlanmasını beklemelidir.

Future, tamamlanmış veya tamamlanmamış olarak iki durumda olabilir. Asenkron bir fonksiyon çağırıldığında tamamlanmamış bir future değeri döner. Bu future, asenkron işlemlerin tamamlanmasını bekler.

Bir future tamamlandığında hangi veri tipine ait bir değer döndürecek ise bu future değerini döndürmek için kullanılan fonksiyonun başında bunu belirtmekte fayda vardır.

Bizim örneğimizde siparisHazirla fonksiyonumuzdaki future bir string değeri döndürdüğü için siparisHazirla fonksiyonunun başında bu belirtilebilir.

```
Future<String> siparisHazirla(){

  return Future.delayed(Duration(seconds:4), (){return 'Siparişiniz alındı';});
```

İlgili fonksiyon, future geriye bir tamsayı döndürüyorsa Future<int>, future geriye bir değer döndürmüyor içinde sadece komutlar çalışıyorsa Future<void> olarak tanımlanabilir.

Future döndüren fonksiyonlarla çalışmak ve onları kontrol etmek için await ve asycn kullanmamız gerekir.

siparisVer fonksiyonu içerisinde, kullanılan siparisHazirla fonksiyonunun işlemini tamamlamasını beklemek için siparisHazirla fonksiyonun başına ‘await’ getirilir.

```
var siparisDurumu= await siparisHazirla();
```

‘await’ sadece asenkron fonksiyonlarda kullanılabilir. Bu nedenle siparisVer fonksiyonunun da asenkron yapılması gerekir. ‘async’ ifadesi ile asenkron yapılabilir.

SiparisVer fonsiyonu artık asenkron bir fonksiyon olduğu için ve geriye herhangi bir değer de döndürmediği için başına Future<void> eklenebilir.

Programın son durumu;

```
void main() {
  siparisVer();
}
Future<void> siparisVer() async {
  var siparisDurumu = await siparisHazirla();
  print('Sipariş Durumu = $siparisDurumu');
}
Future<String> siparisHazirla(){
  return Future.delayed(Duration(seconds:4), (){return 'Siparişiniz alındı';});
  //return Future.delayed(Duration(seconds:4), ()=>'Siparişiniz alındı');
}
```

STREAM :

```
int a=5          //  Tekil       List a=[1,2,5]   //  Çoğul
Future<int>      // Tekil        Stream           // İçinde bir çok Future tutan bir liste gibidir.
```

LİSTELEME // forEach() , where() ve map() metodları

forEach(): void forEach(void Function(int) action)

Listenin elemanlarını sırasıyla fonksiyona parametre olarak gönderir. Bu fonksiyon parametre alan ama değer döndürmeyen bir fonksiyondur.

Anonim bir fonksiyon kullanarak bir örnek;

```
void main() {
 List isim=['Ali','Mehmet'];
  isim.forEach((parametre)=>print(parametre));
 // isim.forEach((parametre) { print (parametre); });
}
```

```
Ali
Mehmet
```

Harici bir fonksiyon kullanarak bir örnek;

```
void main() {
  List isim = ['Ali','Mehmet'];
  isim.forEach(yazdir);
}
yazdir(parametre) {
  print (parametre);
}
```

```
Ali
Mehmet
```

where(): Iterable<int> where(bool Function(int) test)

Bu metod listenin her bir elemanını fonksiyona gönderir. Bu metod parametre olarak bir fonksiyon alır. Bu fonksiyon bir tamsayı parametresi almalıdır ve bir boolean değeri döndürmelidir.

Where metodu ile verilen bir listeden şarta uyanlar alınıp yeni bir liste oluşturulur.

```
void main() {
   List<int> a=[1,2,3,4,5,6,7,8,9];
  print(a.where((int x)=>x<3));                    //   (1, 2)
}
```

Where metodu a listesindeki tüm elemanları tek tek anonim fonksiyona gönderir. Elemanların, fonksiyon içindeki şarta uyup uymadıkları kontrol edilir. Şarta uyanlar, where metodu ile **iterable tipinde** bir koleksiyon olarak döndürülür.

Dönecek değerleri bir değişkene de alınabilir.

```
void main() {
  List<int> a=[1,2,3,4,5,6,7,8,9];
  var bulunanlar=a.where((int x)=>x<3);
  print(bulunanlar);                                 //    (1, 2)
}
```

Where metodu ile dönen iterable tipindeki koleksiyon listeye (**list veri tipine**) çevrilip kullanılabilir.

```
void main() {
 List<int> a=[1,2,3,4,5,6,7,8,9];
 var bulunanlar=a.where((int x)=>x<3).toList();
 print(bulunanlar);                                    //     [1, 2]
}
```

Örnekteki anonim fonksiyon, ayrı bir fonksiyon olarak da yazılabilir;

```
void main() {
 List<int> a = [1, 2, 3, 4, 5, 6, 7, 8, 9];
 var bulunanlar = a.where(kontrol).toList();
 print(bulunanlar);                                  //    [1, 2]
}

bool kontrol(int sayi) {
 if (sayi < 3) {
  return true;
 } else {
  return false;
 }
}
```

```
bool kontrol(int sayi) {
 return sayi<3;
}
```
Fonksiyon bu şekilde de yazılabilir.

Map(): Iterable<T> map<T>(T Function(int) toElement)

Map metoduna, tamsayı parametresi alan bir fonksiyon verilir. Listenin elemanları tek tek bu fonksiyona gönderilir. Her bir eleman, gönderilen fonksiyondaki işlemlerden geçer. Sonuç olarak geriye bir iterable döndürür.

```
void main() {
 List<int> a = [1, 2, 3, 4, 5, 6, 7, 8, 9];
 print(a.map(donustur));                    //   (2, 3, 4, 5, 6, 7, 8, 9, 10)
}
donustur(int sayi) {
 return sayi+1;
}
```

Donuştur() fonksiyonunda return ile ne belirlenirse elemanlar o değere dönüşmüş olur;

```
void main() {
 List<int> a = [1, 2, 3, 4, 5, 6, 7, 8, 9];
 print(a.map(donustur));                    //   (5, 5, 5, 5, 5, 5, 5, 5, 5)
}
donustur(int sayi) {
 return 5;
}
```