

```

int[,] d = {{2,3},{4,5},{6,7}}; // 3x2 lik int dizi.
d[2,1]=33; //doğru
d[2][1]=33; //hata
Console.WriteLine(d.GetValue(2,1)); // 7 değerini yazacak
int[][] dd = new int[2][]; // şekildeki düzensiz dizi oluşturuluyor.ilk önce satırları
dd[0] = new int[2]; //ilk satırın 2 sütunlu olacağı
dd[1] = new int[1]; //ikinci satırın 1 sütunlu olacağı
dd[0][0] = 23;
dd[0][1] = 234;
dd[1][0] = 44;
Random rnd = new Random();
int[] dizi =new int[3];
dizi[0] = rnd.Next(2,10); // 2 ile 10 arası rastgele sayı
dizi[1] = rnd.Next(50); //0 ile 50 arası
dizi[2] = rnd.Next(); //rastgele integer değer
Console.WriteLine(dizi.GetValue(2));//dizi değişkeninin 2. indis elemanını (3.eleman) yazar.
Console.WriteLine(dd[1][0]);
string[] isimler ={ "ali", "ahmet", "selda", "canan", "melike" };
Console.WriteLine("aranan isim=");
string aranan=Console.ReadLine();
aranan.ToLower();
foreach(ss in isimler) {
    if (aranan.Equals(ss)
        Console.WriteLine("aranan isim bulundu...");
        else Console.WriteLine("isim yok");
    }
    Console.ReadKey();

```

23	234
44	

# Metotlar ve Fonksiyonlar

- *Metot Nedir?*
- *Metot bildirimi*
- *Metot Parametresi olarak diziler*
- *Değer ve Referans Parametreleri*
- *Ref ve out Anahtar sözcükleri*
- *Recursive (Özyineli) metotlar*
- *Main, Math Sınıfı*

# Metotlar

*Programlarımızda iş yapan en temel parçalar fonksiyonlardır.*

*Metot bildirimi ;*

```
[erişim] <dönüş değeri> metot ismi (parametre listesi)
{
metot gövdesi;
}
```

# Metotlar

*Bildirimdeki [erişim] seçeneği metoda nasıl ulaşılacağını gösterir. Eğer bir ifade yazılmazsa metot Private kabul edilir.*

```
int IlkMetot(int a,int b)
{
return (a+b);
}
```

**C#** %100 nesne yönelimli bir dil olduğundan, bir metot kullanılırken bulunduğu sınıf türünden bir nesne tanımlanır ve “.” operatörü ile metoda ulaşılır.

# Metotlar

*Geri dönüş değeri olmayan metotlar “void” olarak tanımlanır. “void” tanımlı bir fonksiyon için “return” kullanılamaz.*

*Eğer giriş parametresi yoksa parantez içi boş bırakılır.*

*void yaz()*

*{*

*Console.WriteLine(“kocaeli”);*

*return; //hata*

*}*

# Metotlar

*'static' olarak tanımlanan metotları çağırmak için bir nesne tanımlamaya gerek yoktur. Eğer metot, içinde olduğu sınıftan çağrılacaksa metodun sadece ismi yazılarak çalıştırılabilir, eğer sınıf dışından bir çağrı yapılacaksa "**SınıfAdi.Metot()**" şeklinde çağrılmalıdır.*

*Static tanımı ile oluşturulmuş metot için new operatörü ile yeni bir nesne tanımlamaya gerek yoktur.*

```
static int Topla(int a,int b)
```

```
{
```

```
    return a+b;
```

```
}
```

```
Topla(4,5);    veya    Sınıf.Topla(5,6);
```

# Metotlar

*Metotların dönüş değeri bir nesne değildir o yüzden aşağıdaki şekilde bir atama söz konusu değildir.*

*Toplam(3,5) = 8; // Geçersiz*

```
using System;
using System.Collections.Generic;
using System.Text;

namespace fun
{
    class Sınıf
    {
        public static void Yaz(int a)
        {
            Console.WriteLine("değer=" + a);
        }
    }
    class Program
    {
        static void Yaz(int a) //Static tanımlama nesne için bir bakıma global bir değer tanımı şeklinde oluyor.
        {
            a += 100;
            Console.WriteLine("değer =" + a);
            return;
        }
        static void Main(string[] args)
        {
            int a=134;
            Sınıf.Yaz(a);
            Yaz(a);
            Console.ReadKey();
        }
    }
}
```



# Metotların Dizilerle kullanımı

*Diziler C# 'ta ayrı bir tür olarak ele alındığı için metotlara dizileri aktarmak normal veri türlerinde olduğu gibidir.*

```
static void DiziYaz(int[] a)
{
    foreach (int i in a)
    {
        Console.WriteLine("dizi=" + i);
    }
}
```

```
static void Main(string[]
args)
{
    int[] aa ={ 3, 4, 5, 7 };
    Sınıf.Yaz(a);
    DiziYaz(aa);
    Console.ReadKey();
}
```

# Değer ve Referans Parametreleri

*Referans türleri metotlara aktarılırken bit bit kopyalanmaz, yani metoda aktarılan sadece bir referanstır. Metot içinde yapılacak değişiklikler nesnenin(dizinin) değerini değiştirebilir.*

*C++ dilinde bir nesnenin metoda aktarımı değer ile (call by value) ve referans (call by reference) ile gerçekleştirilir.*

*C# dilinde değer tipleri metoda bit bit kopyalanır.*

# Değer ve Referans Parametreleri

```
static void Main(string[] args)  
{  
  
    int a=134;  
    Console.WriteLine(a);  
    a=Yaz(a);  
    Console.WriteLine(a);  
    Console.ReadKey();  
}
```

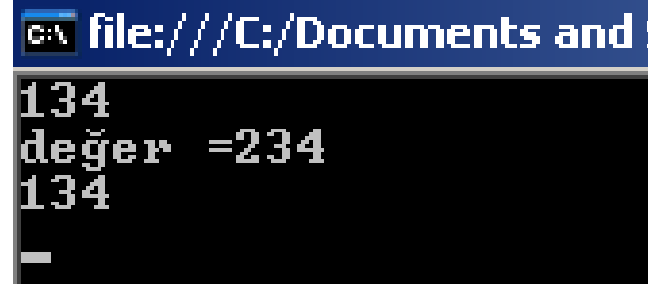
*çıktı*

*// 134*

*11//deger=234*

*//234*

```
static int Yaz(int a)  
{  
    a += 100;  
    Console.WriteLine("değer =" + a);  
    Return a;  
}
```

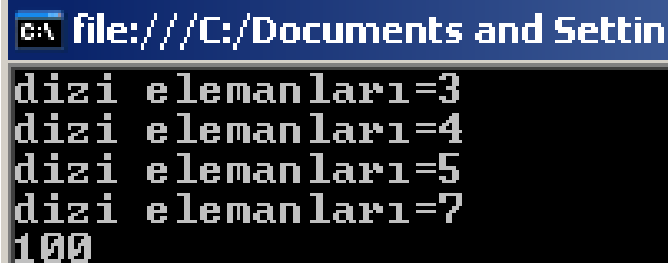


```
file:///C:/Documents and Settings/...  
134  
değer =234  
134  
_
```

# Değer ve Referans Parametreleri

```
static void Main(string[] args)  
{  
  
    int[] aa = { 3, 4, 5, 7 };  
    DiziYaz(aa);  
    Console.WriteLine(aa[0]);  
    Console.ReadKey();  
}
```

```
static void DiziYaz(int[] a)  
{  
    foreach (int i in a)  
    {  
        Console.WriteLine("dizi=" + i);  
    }  
    a[0] = 100;  
}
```



```
file:///C:/Documents and Settings  
dizi elemanları=3  
dizi elemanları=4  
dizi elemanları=5  
dizi elemanları=7  
100
```

# Ref ve out anahtar sözcükleri

*ref anahtarı metotlara parametre aktarırken kullanılır. ref anahtar sözcüğü ile belirtilen parametreler değer tipi de olsa referans olarak aktarılırlar.*

```
static void DenRef(ref int x)
{
    x=50;
}
```

```
static void Main()
{
    int x=10;
    DenRef(ref x);
    Console.Write(x);
}
```

# Ref ve out anahtar sözcükleri

*İlk değer ataması yapılmamış değişkenleri **ref** anahtar sözcüğü ile metodlara parametre olarak aktaramayız. Bu tür hataları engellemek ve bu kullanımı geçerli kılmak için **out** anahtar sözcüğü kullanılır. Ref ten tek farkı ilk değer atamasına gerek yok.*

```
static void DenOut(out int x)
{
    x=50;
}
```

```
static void Main()
{
    int x;
    DenOut(out x);
    Console.Write(x);
}
```

# Metotların aşırı yüklenmesi

*Aynı isme sahip metotların çağırılması çalışma zamanında metotların imzalarına (method signature) bakılarak yapılır.*

```
static void Main(string[] args)
{
    topla(2.45f, 4.5f);
    topla(3, 4,5);
    topla('a', 'c');
    Console.ReadKey();
}
```

```
static void topla(int a, int b,c,d,e)
{ Console.WriteLine(a + b);
}
static void topla(float a, float b)
{ Console.WriteLine(a + b);
}
static void topla(char a, char b)
{
    Console.WriteLine(Convert.ToStri
ng(a) + b);
}
```

# Metotların aşırı yüklenmesi

*Fakat **topla(12.34,1.45)** şeklinde bir çağrım söz konusu ise , burada gönderilen parametreler Double tipinde olduğu için herhangi bir tipe uymaz double tipide int veya char veya float'a dönüşmeyeceği için derleyici hata verir.*



## Değişken sayıda Parametre(params anahtarı)

*Şimdiye kadar metodlarımızda belli sayıda parametre almaktaydı. Fakat bazen metotlara göndereceğimiz parametre sayısını önceden kestiremeyebiliriz.*

*Bu durumda **params** anahtar sözcüğü kullanılır. params anahtarı değişken sayıda eleman içerebilen bir veri yapısıdır.*

## Değişken sayıda Parametre(params anahtarı)

```
static int Toplam(params int[]  
sayi)  
{  
    if (sayi.Length == 0)  
        return 1;  
    int top = 0;  
    foreach (int s in sayi)  
        top += s;  
    return top;  
}
```

```
static void Main()  
{  
    Console.WriteLine(Toplam());  
    Console.WriteLine( Toplam(3, 4));  
    Console.ReadKey();  
}
```

# Recursive metotlar

```
static int fakt(int a)  
{  
    if (a<2) return 1;  
    return a*fakt(a-1);  
}  
Static void Main()  
{  
    Console.WriteLine(fakt(10));  
}
```

# Haftaya nesneler ve sınıflar