EMM3115
İLERİ PROGRAMLAMA – C#
KONTROL YAPILARI, DÖNGÜLER, DİZİLER, METOTLAR

Doç. Dr. İbrahim KÜÇÜKKOÇ

Web: http://ikucukkoc.baun.edu.tr

Email: ikucukkoc@balikesir.edu.tr

Yararlanılan Kaynaklar:
Geleceği Yazanlar Kulübü - https://gelecegiyazanlar.turkcell.com.tr/konu/c-sharp/egitim/101
C# Eğitim Kitabı 2. Baskı, Murat Yücedağ, Dikeyeksen Yayınları, 2019
https://www.kodyaz.net/

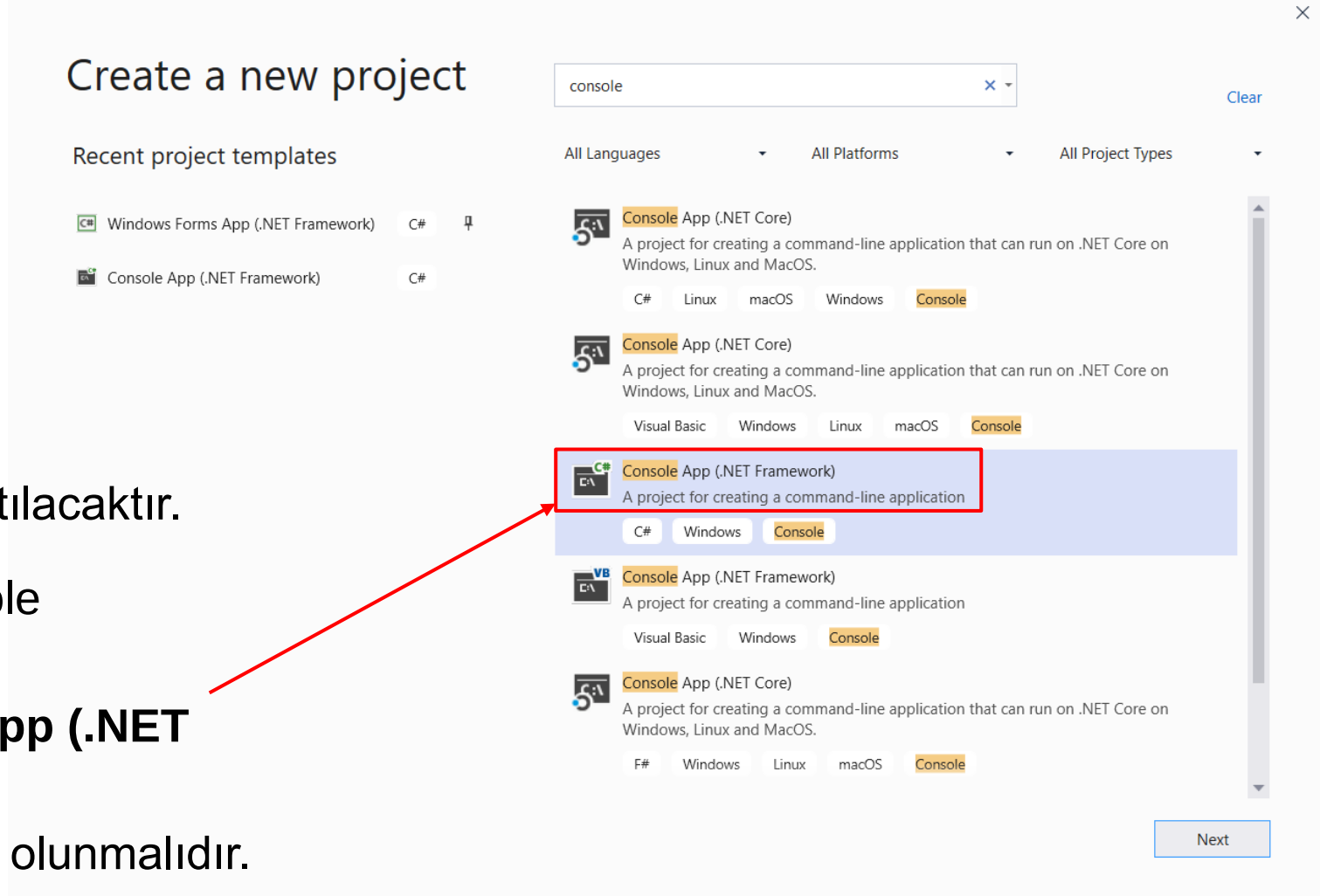
Güncelleme: 31 Temmuz 2020

Başlamadan Önce…

- Bu slaytta sırasıyla;
  - Kontrol Yapıları
  - Döngüler
  - Diziler ve
  - Metotlar

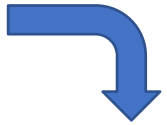
Console uygulaması üzerinden anlatılacaktır.

- Visual Studio 2019'da yeni bir Console uygulama projesi oluşturmak için **File -> New -> Project ->Console App (.NET Framework)** seçilmelidir.
  Uyarı: C# dili için seçildiğinden emin olunmalıdır.



- Eğer **Windows Forms Application** üzerinden çalışılmak istenirse, bu slaytta verilecek kodlar uygun şekilde değiştirilerek aynı örnekler Windows Forms Application üzerinden de yapılabilir.
- Bunun için örneğin **Console.ReadLine()** komutu ile konsoldan alınacak bir değer, bir butona basıldığında **textBox1.Text()** ile alınabilir.
- Benzer şekilde **Console.WriteLine()** komutu ile konsolda gösterilecek olan bir sonuç bir butona basıldığında **MessageBox.Show(a)** komutu ile gösterilebilir.

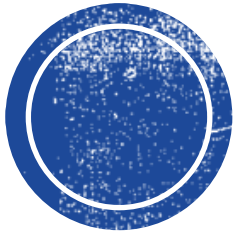
```
int a = Convert.ToInt32(Console.ReadLine());
```



```
int a = Convert.ToInt32(textBox1.Text());
```

```
Console.WriteLine(a);
```

```
MessageBox.Show(a);
```

DEĞİŞKENLER

Bu bölümde C# programlama dilinde kullanılan değişkenler Console uygulamaları üzerinden anlatılacaktır.

Sayısal Veri Tipleri

Değişken Tipi	Bellekte Kapladığı Alan	Alabileceği Değer Aralığı
byte	1 Byte	0 ile 255 arasındaki tam sayı değerleri
sbyte	1 Byte	-128 ile 127 arasındaki tam sayı değerleri
short	2 Byte	-32768 ile 32767 Aralığındaki tam sayı değerleri
ushort	2 Byte	0 ile 65535 Aralığındaki tam sayı değerleri
int	4 Byte	-2.147.483.648 ile 2.147.483.647 Aralığındaki tam sayı değerleri
uint	4 bayt	0 ile 4.294.967.295 Aralığındaki tam sayı değerleri
long	8 bayt	-9.223.372.036.854.775.808 ile 9.223.372.036.854.775.807 Aralığındaki tam sayı değerleri
ulong	8 bayt	0 ile 18.446.744.073.709.551.615 Aralığındaki tam sayı değerleri
float	4 bayt	$\pm 1.5*10^{-45}$ ile $\pm 3,4*10^{38}$ Aralığındaki reel sayılar
double	8 bayt	$\pm 5.0*10^{-324}$ ile $\pm 1,7*10^{308}$ Aralığındaki reel sayılar
decimal	16 bayt	$\pm 1.5*10^{-28}$ ile $\pm 7,9*10^{28}$ Aralığındaki reel sayılar

Alfasayısal Veri Tipleri

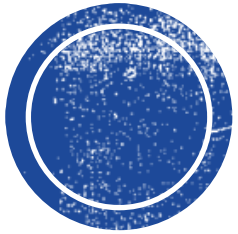
Değişken Tipi	Bellekte Kapladığı Alan	Alabileceği Değer Aralığı
Char	2 Byte	Tek karakterlik bir ifade tutar.
String	Sınırsız	Metinsel bir ifade tutar. Unicode karakterlerden oluşur.

Diğer Veri Tipleri

Değişken Tipi	Bellekte Kapladığı Alan	Alabileceği Değer Aralığı
Bool	2 Byte	Mantıksal veri tipi olup, True ya da False değerlerini alabilir.
Object	Sınırsız	Tüm verileri kapsar. Zaten tüm veri tipleri bu nesneden türemiştir.
DateTime	8 Byte	Tarih ve zaman bilgisini tutar.

Kaçış Karakterleri

Karakter	Açıklama
\n	Yeni satıra geçmek için kullanılır.
\t	Tab tuşu işlevini gerçekleştirir.
\0	Null
\"	Çift tırnak (") karakterinin ekrana basılması
\r	Satır başına geçmek için kullanılır.
\b	Geri al
\\	'\' karakterini ekrana basabilmek için kullanılır.
\a	Uyarı (bip) sesi



KONTROL YAPILARI

Bu bölümde C# programlama dilinde kullanılan IF ve SWITCH deyimleri Console uygulamaları üzerinden anlatılacaktır.

Kontrol Yapıları

- **İlişkisel ve Mantıksal Operatörler**
- Uygulama geliştirilirken, programın akışı sırasında verilmesi gereken çeşitli kararlar için belirli yapıların ve ölçütlerin tanımlanması gerekir. Verilecek bu kararlara göre programın akışı farklı bir yönde ilerletilebilir ve her çalıştırılmasında o anki duruma uygun sonuçlar vermesi sağlanabilir. Bu karar yapıları, belirli karşılaştırmaların yapılmasını da gerektirebilir.
- Örneğin, çeşitli nedenlerle kullanıcıdan alınan bir bilgiyi belirli bir ölçüte göre sınamak isteyebilirsiniz. Kullanıcıdan alınan bir değerin sıfıra eşit olup olmadığını kontrol etmek, daha sonradan bu değeri bir bölme işleminin paydası olarak kullanmanız durumunda ortaya çıkabilecek mantıksal hata gibi problemlerin önüne geçmenizi sağlayabilir.

- **İlişkisel Operatörler:**

==	Eşittir
!=	Eşit değildir
<	Küçüktür
>	Büyüktür
>=	Büyüktür veya eşittir
<=	Küçüktür veya eşittir

- **Mantıksal Operatörler:**

&	VE
|	VEYA
||	Kısa Devre VEYA
&&	Kısa Devre VE
!	DEĞİL

- İlişkisel ve mantıksal operatörler doğru (**true**) yada yanlış (**false**) değerini, yani bool tipinde bir değeri geriye döndürür ve çoğunlukla **if** anahtar kelimesi ile birlikte kullanılır.
- Mantıksal operatörler ise iki bool değerini karşılaştırmak için kullanılır.
- Mantıksal Operatörlerin Sonuçları:

p	q	p & q	p | q	!p
false	false	false	false	true
true	false	false	true	false
false	true	false	true	true
true	true	true	true	false

- Kısa devre operatörleri, normal VE (**&**) ve VEYA (**|**) operatörlerinden biraz daha farklı çalışır. Normal operatörlerin kullanıldığı bir deyimde, ilk operandın sonucu ne olursa olsun diğer operandlar da kontrol edilir. Örneğin, normal VE operatörüyle yapılan bir işlemde, ilk operandın sonucu yanlış (**false**) ise, ikinci operandın sonucu ne olursa olsun, sonuç yanlış (**false**) olacaktır.
- Aynı şekilde, VEYA (**|**) operatörü kullanıldığında, ilk operandın sonucu doğru (**true**) ise, ikinci operandın sonucu ne olursa olsun, sonuç doğru (**true**) olacaktır. Bu iki durumda da, ikinci operandın kontrol edilmesi gereksizdir, ancak normal operatörlerde bu kontrol yapılır. Kısa devre operatörleri (**&&** ile **||**) kullanıldığında ise, ilk operandla alınan sonuca bağlı olarak, ikinci operand kontrol edilmez.

- Kısa devre VE (**&&**) işleminde ilk operandın yanlış (**false**), kısa devre VEYA (**||**) işleminde ilk operandın doğru (**true**) olması durumunda ikinci operand kontrol edilmez. Bu, performans artışı sağlayacağı gibi, bazı istisnai durumlarda programın hata vermesini de engeller.
- **int toplam = x + y ;** işleminde**, *x*** ve ***y*** ifadeleri birer **operand**, **+** ise bir **operatördür.**

IF Koşulu

- C#'ta uygulama akışlarını kontrol etmek için, genel olarak tercih edilen iki adet deyim vardır. Bunlardan biri **if**, diğeri ise **switch** ifadesidir.
- **if**, uygulamada bir koşulun kontrolünü sağlayan deyimdir. Genel kullanımı aşağıdaki gibidir:

```
if (koşul)
{
    Koşul doğru ise çalıştırılacak komutlar;
}
else
{
    Koşul yanlış ise çalıştırılacak komutlar;
}
```


- İlişkisel ve mantıksal operatörler kullanılarak, **bool** değerler döndüren koşullar oluşturulabilir ve kontrol sağlanabilir.
- **if** deyimi koşul doğru ise kendi altındaki kod bloğunu, doğru değilse **else** anahtar sözcüğünün altındaki kod bloğunu çalıştırır.
- **if** veya **else** deyiminden sonra sadece tek satır yazılacaksa **{** işaretine gerek yoktur. Aslında bu kural, C#'taki birçok deyim için geçerlidir.
- Aşağıda, kullanıcının girdiği parolayı kontrol eden örnek bir konsol uygulaması görülüyor:

```
string parola = Console.ReadLine();
if (parola == "1234")
{
    Console.WriteLine("Parola doğru");
}
else
{
    Console.WriteLine("Girilen parola doğru değil");
}
```

Çok Koşullu if Yapısı

- if deyimi birden fazla durumlu koşulların sınanması amacıyla da kullanılabilir. Bu amaçla **else if** (koşul) yapısı kullanılır. Bu duruma ait kullanılacak yapı ise aşağıdaki gibidir:

```
if (koşul1)
{
    Koşul1 doğru ise çalıştırılacak komutlar;
}
else if (koşul2)
{
    Koşul1 yanlış ve Koşul2 doğru ise çalıştırılacak komutlar;
}
else
{
    Koşul1 ve Koşul2 yanlış ise çalıştırılacak komutlar;
}
```


İç İçe if Yapısı

- if deyimi iç içe olarak da kullanılabilir. Bu yapıda parantezlerin doğru bir şekilde kullanılmasına dikkat edilmelidir. Bu duruma ait kullanılacak yapı aşağıdaki gibidir:

```
if (koşul1)
{
    Koşul1 doğru ise çalıştırılacak komutlar;
    if (koşul2)
    {
        Koşul1 ve Koşul2 doğru ise çalıştırılacak komutlar;
    }
    else if (koşul3)
    {
        Koşul1 ve Koşul3 doğru, Koşul2 yanlış ise çalıştırılacak komutlar;
    }
}
else if (koşul4)
{
    Koşul1 yanlış ve Koşul4 doğru ise çalıştırılacak komutlar;
}
else
{
    Koşul1 ve Koşul4 yanlış ise çalıştırılacak komutlar;
}
```

- Şimdi, örneği biraz geliştirelim. Kullanıcıdan parola dışında bir de kullanıcı adı bilgisi isteyelim ve her ikisi de doğru ise işlem başarılı olsun:

```
string kullaniciAdi = Console.ReadLine();
string parola = Console.ReadLine();

if (kullaniciAdi == "admin" && parola == "1234")
{
    Console.WriteLine("Kullanıcı Adı ve Parola Doğru");
}
else
{
    Console.WriteLine(" Kullanıcı Adı ve Parola Yanlış!");
}
```

- Burada **if** deyiminde **VE (&&)** operatörü ile iki koşulun da **true** (doğru) olması durumunda giriş doğrulanır. Fakat durumun **false** (yanlış) olmasının hangi koşuldan kaynaklandığı bilinemez. Bu durum bir sonraki slayttaki gibi iç içe **if** deyimleri kullanılarak çözülebilir.

```
string kullaniciAdi = Console.ReadLine();
string parola = Console.ReadLine();

if (kullaniciAdi == "admin")
{
    if (parola == "123456")
    {
        Console.WriteLine("Kullanıcı Adı ve Parola Doğru");
    }
    else
    {
        Console.WriteLine("Girilen Parola Doğru Değil!");
    }
}
else
{
    Console.WriteLine("Girilen Kullanıcı Adı Doğru Değil!");
}
```

SWITCH Deyimi

- **switch** deyimi de, **if** deyimi gibi uygulama akışını kontrol etmek için kullanılır. Genellikle daha okunaklı ve birçok durum için daha verimli olduğu için, karmaşık **else if** blokları yerine kullanılır. Genel yazım şekli şöyledir:

```
switch (değişken)
{
    case 1:
        //değişken değeri 1’e eşitse yapılacak işler
    break;
    case 2:
        // değişken değeri 2’ye eşitse yapılacak işler
    break;
    case 3:
        // değişken değeri 3’e eşitse yapılacak işler
    break;
    default:
        // değişken değeri yukarıdakilerin hiç birine eşit değilse yapılacak işler;
    break;
}
```

Sık Yapılan Hatalar

- **IF Deyimiyle İlgili Sık Yapılan Hatalar**
  - if bloklarının sonunda “;” kullanılması. if ();
  - if bloğunun sonunu belirleyen küme parantezlerinden sonra “;” kullanmak {};
  - Çok satırlı if kullanımında { } parantezlerinin kullanılmaması durumunda sadece bir satır işletilir.
  - Else bloğundan sonra “;” kullanılması. (Else;)
  - if bloğunda karşılaştırma operatörü olarak “==” yerine “=” kullanmak.
  - Else if bloğunu bitişik yazmak. Elseif()
- **Switch Deyimiyle İlgili Sık Yapılan Hatalar**
  - Break komutu unutulursa koşula bakılmaksızın diğer case bloğu işletilir.
  - Switch (Koşul); ifadesinin sonuna “;” koyulması.
  - Case satırının sonuna “;” koyulmaması.
  - Case etiketinin birleşik yazılması. CaseDeger;

Örnek Uygulama-1

- Kullanıcıdan alınan bir sayının çift olup olmadığını tespit edip ekrana yazdıran C# programını geliştiriniz.

```
Console.WriteLine("Bir sayı giriniz: ");
int a = Convert.ToInt32(Console.ReadLine());
if (a % 2 == 0)
{
    Console.WriteLine("Girilen Sayı Çifttir");
} else
{
    Console.WriteLine("Girilen Sayı Çift DEĞİLDİR");
}
```


Örnek Uygulama-2

- Girilen iki sayıdan büyük olanı bulup ekrana yazan C# uygulamasını yazınız.

```
int sayi1, sayi2;
Console.Write("Birinci sayıyı girin.");
sayi1 = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("İkinci sayıyı girin.");
sayi2 = Convert.ToInt32(Console.ReadLine());
if(sayi1>sayi2)
    Console.WriteLine("1. sayı olan {0} daha büyük.",sayi1);
else
    Console.WriteLine("2. sayı olan {0} daha büyük.", sayi2);

//Çalışma hatasını bulun?
Console.ReadKey();
```


Örnek Uygulama-3

- Kullanıcıdan alınan 2 sayıya yine kullanıcıdan alınacak değere göre dört işlem uygulayan C# programını yazınız. (Toplama için 1, Çıkartma için 2, Çarpma için 3, Bölme için 4 girilecek. Bu değerler dışındaki girişler için hatalı giriş yaptınız şeklinde uyarı verecek.)

```
int sayi1, sayi2, sonuc;
byte islem; //Neden byte tanımladık ?
Console.Write("Birinci sayıyı girin."); //Write ve WriteLine farkına
dikkat edin.
sayi1 = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("İkinci sayıyı girin.");
sayi2 = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("İkinci sayıyı girin.");
Console.WriteLine("Yapılacak olan işlemi seçin.");
islem = Convert.ToByte(Console.ReadLine());
```

...

(devamı sonraki slaytta)

(devamı)

```
if (islem == 1)
{
 sonuc = sayi1 + sayi2;
 Console.WriteLine("İşlem sonucu={0}.",sonuc);
 //"{0}" kullanımına dikkat;
}
else if (islem == 2)
{
 sonuc = sayi1 - sayi2;
 Console.WriteLine("İşlem sonucu={0}.", sonuc);
 //"{0}" kullanımına dikkat;
}
else if (islem == 3)
{
 sonuc = sayi1 * sayi2;
 Console.WriteLine("İşlem sonucu={0}.", sonuc);
}
else if (islem == 4)
{
 sonuc = sayi1 / sayi2;
 Console.WriteLine("İşlem sonucu={0}.", sonuc);
}
else
{
 Console.WriteLine("Hatalı değer girdiniz. ");
}
//Kod kısaltma nerelerde yapılabilir?
```

Örnek Uygulama-4

- Bir mağaza müşterilerine yaptıkları alışveriş tutarına göre indirim yapmaktadır.
  200 TL ye kadar olan alışverişler için %10
  200-400 TL arası olan alışverişler için %15
  400 TL den fazla olan alışverişler için %20 Buna göre bir kişinin ödeyeceği net miktarı hesaplayan C# programın kodunu yazınız.

```
int alisverisTutari,indirimliAlisveriTutari;
byte indirimOrani=0; //Neden 0 tanımladık ?
Console.Write("Alışveriş tutarını girin.");
alisverisTutari = Convert.ToInt32(Console.ReadLine());
if(alisverisTutari<=200)
 indirimOrani = 10;
else if(alisverisTutari<=400)
 indirimOrani = 15;
else if(alisverisTutari>400)
 indirimOrani = 20;

indirimliAlisveriTutari = alisverisTutari - alisverisTutari * indirimOrani / 100;
Console.Write("Alışveriş tutarınız={0} ve indirim oranınız= {1}, indirimli alışveriş
tutarınız={2}",alisverisTutari,indirimOrani,indirimliAlisveriTutari);
//{} kullanımına dikkat
Console.ReadLine();
```

Örnek Uygulama-5

- Bir otoparkın ücret tarifesi şöyledir:

  0 – 3 saat: 4 TL

  3 – 7 saat: 3 TL

  7 – 12 saat: 2 TL

  12 ve üzeri: 1 TL'dir.

  Buna göre girilen saate göre otoparka ödenecek ücreti hesaplayıp ekrana yazan programı oluşturunuz.

```
int parkSaati=0;int ucret=0;
Console.Write("Park süresini girin.");
parkSaati = Convert.ToInt32(Console.ReadLine());
if (parkSaati < 3)
   ucret = parkSaati * 4;
else if(parkSaati>3 && parkSaati<7)
   ucret = parkSaati * 3;
else if (parkSaati >= 7 && parkSaati < 12)
   ucret = parkSaati * 2;
else if (parkSaati>=12)
   ucret = parkSaati * 1;
Console.Write("Ödenecek Toplam Park Ücreti = {0}.",ucret);
```

Örnek Uygulama-6

- İlk olarak ekranda

  1-Kare

  2-Dikdörtgen

gibi iki seçenek görünür. Kullanıcı hangi seçeneği seçerse seçilen seçeneğe göre

1-Alan

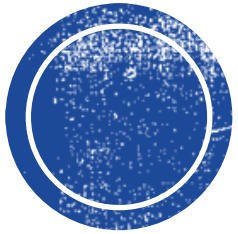
2-Çevre

seçenekleri ekrana gelir. Hangi seçimi yaparsa onunla ilgili bilgi alma ekranı gelir ve sonuç görüntülenir.

```
string kareDikdortgen = "", alanCevre = "";
int a, b, cevre, alan;
Console.WriteLine("1- Kare");
Console.WriteLine("2- Diktörtgen");
Console.Write("Seçiminiz (1-2) : ");
kareDikdortgen = Console.ReadLine();
if (kareDikdortgen == "1")
{
  Console.WriteLine("1- Alan");
  Console.WriteLine("2- Çevre");
  Console.Write("Seçiminiz (1-2) : ");
  alanCevre = Console.ReadLine();
  if (alanCevre == "1")
    {
       Console.Write("Kenarı giriniz = ");
       a = int.Parse(Console.ReadLine());
       alan = a * a;
       Console.WriteLine("Alan = {0}", alan);
    }
  if (alanCevre == "2")
    {
       Console.Write("Kenarı giriniz = ");
       a = int.Parse(Console.ReadLine());
       cevre = 4 * a;
       Console.WriteLine("Çevre={0}", cevre);
    }
}
```

(devamı)

```
if (kareDikdortgen == "2")
{
  Console.WriteLine("1- Alan");
  Console.WriteLine("2- Çevre");
  Console.Write("Seçiminiz (1-2) : ");
  alanCevre = Console.ReadLine();
  if (alanCevre == "1")
    {
      Console.Write("Uzun kenarı giriniz = ");
      a = int.Parse(Console.ReadLine());
      Console.Write("Kısa kenarı giriniz = ");
      b = int.Parse(Console.ReadLine());
      alan = a * b;
      Console.WriteLine("Alan = {0}", alan);
    }
  if (alanCevre == "2")
    {
      Console.Write("Uzun kenarı giriniz = ");
      a = int.Parse(Console.ReadLine());
      Console.Write("Kısa kenarı giriniz = ");
      b = int.Parse(Console.ReadLine());
      cevre = 2 * (a + b);
      Console.WriteLine("Çevre = {0}", cevre);
    }
}
```



DÖNGÜLER

Bu bölümde C# programlama dilinde kullanılan FOR, WHILE, DO WHILE ve FOREACH döngüleri Console uygulamaları üzerinden anlatılacaktır.

DÖNGÜLER

- Döngüler, programlama dillerinde en çok ihtiyaç duyulan ifadelerin arasında yer alır. Program akışında tekrar tekrar gerçekleştirilmesi gereken iş süreçleri varsa, bu iş süreçleri döngüler yardımıyla gerçekleştirilir. C# dilinde 4 çeşit döngü vardır:
  - **for** döngüsü
  - **while** döngüsü
  - **do while** döngüsü
  - **foreach** döngüsü

BREAK komutu içinde bulunduğu döngüyü kırar, program kırılan döngüden sonra kaldığı yerden çalışmaya devam eder.

CONTINUE komutu BREAK komutuna benzer. Ancak break komutundan farklı olarak program CONTINUE komutunu gördüğünde döngüden çıkmaz, sadece döngünün o anki iterasyonu sonlanır ve döngü bir sonraki iterasyonu yapmak üzere tekrar başlatılır.

FOR Döngüsü

- **for** döngüsü kullanımında, döngüde kullanılması için bir değişken oluşturulur ve buna bir başlangıç değeri verilir, örnegin **int i = 0** gibi.
- Ardından döngünün sınırlarını belirleyen koşul ifadesi gelir, örneğin **i < 10** ifadesi bir koşuldur.
- Son olarak döngü her çalıştığında değişken üzerinde gerçekleşecek matematiksel ifade yer alır, örneğin **i++** ifadesi gibi. Bu bir sayaç yapısı gibi de düşünülebilir. Gerekli koşul doğru (**true**) olduğu sürece, başlangıç değerine artırım İşlemi gerçekleşir. ***for*** döngüsünün genel yazım şekli aşağıdaki gibidir:

```
for (int i = 0; i < 10; i++)
{
   //döngü çalıştığı sürece çalıştırılacak komutlar
   Console.WriteLine(i);
}
```


- Başlangıç değeri (i), kontrol değişkeni olarak da adlandırılır. Kontrol değişkeni, döngü içinde bir sayaç görevi görür ve genellikle, koşul içinde kontrol edilecek değer olarak kullanılır.
- Koşul, döngünün her seferinde yapılmak istenen operasyonun devam edip etmeyeceğini denetleyen mekanizmadır. Burada genellikle başlangıç değerinin durumu operatörler ile denetlenir.
- Matematiksel işlem, başlangıç değerinin (i) döngü her gerçekleştiğinde üzerinde uygulanacak işlemi işaret eder. Koşul içinde kontrolü sağlanan başlangıç değeri üzerinde bu işlem yapılmazsa, sonsuz döngü oluşur.

- Şimdi, 0'dan 100'e kadar olan sayıların toplamını bulan bir uygulama geliştirelim:

```
int toplam = 0;
for (int i = 0; i < 100; i++)
{
    toplam += i;
}

Console.WriteLine(toplam);
```

Çıktı: 4950

- **int i = 0** biçimindeki başlangıç ifadesi, for döngüsünün başındaki değişkenin ilk değerini belirler. i değişkeni, döngünün hangi adımda olduğunu belirtir.
- **i < 100** ifadesi, döngünün koşuludur. Koşulun görevi, döngünün devam edip etmeyeceğine karar vermektir. Her adımda koşul kontrol edilir. Koşul doğru (true) değerini verdikçe, döngü devam eder. Buradaki örnekte i değişkeni 100'den küçük ve eşit olduğu sürece döngü çalışmaya devam edecektir.
- **i++** biçimindeki matematiksel işlem, döngünün her adımında i'nin değerini 1 artırır. Burada **i=i+2** yazılması durumunda her döngüde i'nin değeri 2 artırılacaktır. Benzer şekilde **i--** veya **i=i-1** yazılırsa her döngüde i'nin değeri 1 eksiltilecektir.

WHILE Döngüsü

- C#'ta yaygın olarak kullanılan döngülerden biri de **while** döngüsüdür. *for* döngüsünde olduğu gibi, bir koşul sağlandığı sürece dönmeye devam eder. Koşul yanlış (**false**) sonucunu verdiği zaman ise sonlandırılır. Genel yazım şekli şöyledir:

```
while (koşul)
{
      koşul sağlandığı sürece çalıştırılacak komutlar
}
```

- **for** döngüsünde yaptığımız toplama örneğini, bir de **while** döngüsüyle yapalım:

```
int toplam = 0;
int i = 0;
while (i < 100)
{
    toplam += i;
    i++;
}
Console.WriteLine(toplam);
```

Çıktı: 4950

- Örnekte görüldüğü gibi, sayaç değişkeni (**i**) döngü içinde, her adımda 1 artırılır. Böylece sayaç 100 olduğunda, döngü sonlandırılır. Burada koşul, sayaç değişkeninin değerinin 100'den küçük olmasıdır. Koşul doğru (**true**) sonucunu verdikçe döngü işletilir, koşulun yanlış (**false**) değerini verdiği durumdaysa döngü sonlandırılır.
- **while** döngüsü, bu şekilde sırayla yapılacak işlemlerden ziyade, problem çözümünde bir koşulun durumunu sınamak için kullanılır.
- Bir sonraki slaytta kullanıcı tarafından girilen bir sayının kaç basamaklı olduğunu bulmak, **while** döngüsü için daha açıklayıcı bir örnek olacaktır.

- Aşağıdaki işlemde **int sayi** isimli bir değişken oluşturulup içine **Console.ReadLine()** metodundan gelen string değer dönüştürülerek atılmıştır. Çünkü **while** döngüsünde büyüktür (**>**) operatörünü kullanmak için bir sayıya ihtiyacımız vardır ve **int** değişkenler büyüktür operatörünü kullanabileceğimiz sayıyı tutmaktadır:

```
int sayi = Convert.ToInt32(Console.ReadLine());
int basamak = 0;
while (sayi > 0)
{
    basamak++;
    sayi = sayi / 10;
}
Console.WriteLine("Girdiğiniz sayı " + basamak.ToString() + "basamaklıdır.");
```


- Yukarıdaki döngüyü biraz inceleyelim: **while** döngüsü, **sayi** değişkeni 0'dan büyük olduğu sürece çalışmakta ve **sayi** değişkeni her döngüde 10'a bölünmektedir ve her döngü çalıştığında basamak isimli değişken **++**; ile 1 arttırılmaktadır.
  (++ ifadesinin kullanılması o değişkenin değerini 1 arttırır)
- Örnek olarak 50 sayısını ele alalım. **while** döngüsü ilk çalıştığında basamak değişkeni 1 arttırılacak, **sayi** 10 ile bölünecek 5 sayısı elde edilecek ve **sayi** isimli değişkene aktarılacaktır. **sayi** isimli değişken hala 0 sayısından büyük olduğu için **while** döngüsü tekrar çalışacaktır. Bu sefer basamak 1 daha artarak 2 olacak, **sayi** değişkeni yani 5 sayısı 10 bölünecek ve 0.5 sonucu elde edilecektir. Değişkenimiz **int** olduğu için **sayi** değeri 0'a eşitlenecek ve **while** döngüsü sonlanacaktır.
- Elde edilen **basamak** değişkeni **Console.WriteLine()** metodu ile ekrana yazdırılmaktadır. Fakat **basamak** değişkeni int olduğundan dolayı stringle beraber yazdırmak için **ToString()** metodunu kullanarak int sayısını string bir metne dönüştürdükten sonra yazdırılmaktadır.

DO WHILE Döngüsü

- *for* ve *while* döngülerinde koşul, döngü başlamadan önce kontrol edilir.
- **Do while** döngüsünde ise, bu kontrol her döngüden sonra gerçekleştirilir. Operasyon mantığında **do while** döngüsü, koşul ne olursa olsun en az bir kere çalıştırılır.
- Bu döngünün genel yazım şekli aşağıdaki gibidir:

```
do
{
    //döngü çalıştığı sürece çalıştırılacak komutlar
} while (koşul)
```


- Toplama örneğimizi bu kez de **do while** döngüsüyle yapalım:

```
int toplam = 0;
int sayac = 0;
do
{
    sayac++;
    toplam += sayac;
}
while (sayac < 100);
Console.WriteLine("Toplam: " + toplam.ToString()
     + ",Sayaç: " + sayac.ToString());
```

Çıktı: 5050

- Bu döngüde koşul sonradan kontrol edildği için sayac değişkeninin alacağı son değer 100 olacak ve çıktı ise 5050 olacaktır.

FOREACH Döngüsü

- *for* döngüsü gibi yaygın kullanılan bir diğer döngü de **foreach** döngüsüdür. **foreach**, dizi (**array**) ve koleksiyon (**collection**) tabanlı nesnelerin elemanları üzerinden ilerleyen, iterasyon gerçekleştirerek bu elemanlara erişip iş katmanınızı oluşturabileceğiniz bir döngüdür.
- Bu döngünün genel kullanım şekli aşağıdaki gibidir:

```
foreach (tip değişken in koleksiyon)
{
    //döngü çalıştığı sürece çalıştırılacak komutlar
}
```


- foreach yapısını anlamak için bir array (dizi) içindeki tüm sayıları çarpan bir döngü yazalım.

```
int[] sayilar = { 1, 2, 3, 4, 5, 6 };
int carpim = 1;
foreach (int x in sayilar){
    carpim = carpim*x;
}
```

burada *x* yerine *sayılar* dizisindeki her eleman tek tek alınıp bununla çarpım işlemi yapılır. Burada çarpım değerinin sonucu 720 olacaktır.

Örnek Uygulama-1

- 7'den 200'e kadar olan sayıların toplamını bulup sonucu ekranda gösteren C# programını yazın.

```
int toplam = 0;
for (int i = 7; i <= 200; i++)
{
    toplam += i;
}
//Console.WriteLine("Toplam = {0}", toplam);
Console.WriteLine("Toplam = " + toplam);
```


Örnek Uygulama-2

- A'dan Z'ye kadar harfleri ekranda yazdıran programın C#kodunu yazınız.

```
char i;
for (i = 'a'; i <= 'z'; i++)
{
    Console.WriteLine(i);
}
```


Örnek Uygulama-3

- Ekrana çarpım tablosunu yazdıran programı geliştiriniz, örneğin 6x2=12 gibi.

```
int i, j;
for (i = 1; i <= 10; i++)
{
    for (j = 1; j <= 10; j++)
    {
        Console.WriteLine("{0}x{1}={2}", i, j, i * j);
    }
}
```


Örnek Uygulama-4

- Girilen iki sayının arasındaki sayıların toplamını bulan C# programını yazın.

```
int toplam = 0, a, b;
Console.WriteLine("bir sayi girin");
a = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("bir sayi girin");
b = Convert.ToInt32(Console.ReadLine());
if (a>b)
{
    int gecici = a;
    a = b;
    b = gecici;
}
for (int i = a; i <= b; i++)
{
    toplam += i;
}
Console.WriteLine("Toplam = " + toplam);
```

Örnek Uygulama-5

- İstenilen sayı kadar girilen sayıların ortalamasını C# programını yazın.

```
int i, sayi, deger;
double toplam = 0;
Console.WriteLine("kaç sayi gireceksiniz");
deger = Convert.ToInt32(Console.ReadLine());
for (i = 1; i <= deger; i++)
{
    Console.WriteLine(i + ".sayiyi giriniz...");
    sayi = Convert.ToInt32(Console.ReadLine());
    toplam += sayi;
}
toplam /= deger;
Console.WriteLine("ortalama: " + toplam);
```

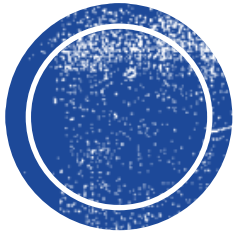

Örnek Uygulama-6: Sayı Tahmin Oyunu

- Bilgisayar tarafından belirlenen rastgele bir sayıyı tahmin edeceğiniz şekilde sizi yönlendiren C# oyun programını yazınız.

```
Console.WriteLine("0 ile 100 arasında bir sayı giriniz");
int sayi=Convert.ToInt32(Console.ReadLine());
int defa = 0; Random rnd = new Random();
int aklimdakiSayi = rnd.Next(0, 100);
do
{
  defa++;
  if (sayi > aklimdakiSayi)
  {
   Console.WriteLine("Daha küçük sayı girin");
   sayi = Convert.ToInt32(Console.ReadLine());
  }
  else if (sayi < aklimdakiSayi)
  {
   Console.WriteLine("Daha büyük bir girin");
   sayi = Convert.ToInt32(Console.ReadLine());
  }
} while (sayi != aklimdakiSayi);
Console.WriteLine("Tebrikler. Sayıyı {0}. Defa da buldunuz.", defa);
```

Çalışma Soruları

- Sayısal loto uygulamasını C# programı kullanarak yapınız. Bu uygulama da 1 ile 49 arasında (sınırlar dahil) rastgele ve birbirinden farklı 7 adet sayı üretilmelidir.
- Kullanıcı tarafından doldurulan 10 boyutlu bir dizideki en büyük ve en küçük değeri dizi fonksiyonlarını kullanmadan bulan ve bunları ekrana yazan C# console uygulamasını yazınız.
- Kullanıcıdan alınan bilgiye göre rastgele şifre üretecek olan programın C# konsole kodunu giriniz. (Örneğin kullanıcı 5 girecek buna uygun olarak rastgele olacak şekilde 5 karakterlik şifre oluşturulacak. Maksimum karakter sayısı 10 olacak ve kullanıcı 10 üzerinde bir değer verirse uyarı verilerek uygun değer girilene kadar kullanıcıdan değer alınacak.)
- Sizin belirlediğiniz bir sayıyı bilgisayar bulacak şekilde programa yönlendirme yapan C# programını yazınız. (Sayıyı siz belirleyeceksiniz ve programınız tahminde bulunacak ve siz klavyeden A(Aşağı) – Y(Yukarı) ile yönlendirme yapacaksınız. T(Tamam) tuşuna basılında program sonlanacak ve kaçıncı defa da bulunduğu ekrana yazılacak.)



DİZİLER

Bu bölümde C# programlama dilinde kullanılan tek ve çok boyutlu diziler Console uygulamaları üzerinden anlatılacaktır.

DİZİLER

- Değişkenler, içlerinde tekil veriler tutan yapılardır. Ancak çok sayıda veri üzerinde çalışmamız gereken durumlarda aynı tipteki değişkenleri bir arada tutmamız gerekebilir. Bu noktada, yazılım dillerinin genelinde kullanılan **dizi (array)** devreye girer.
- Bir dizi tanımlamanın genel yazım şekli şöyledir:

```
tip dizi1 = new tip[elemansayisi];
```

Diziye verilen isim

Dizide tutulacak eleman sayısı

Dizide tutulacak verinin tipi

- Örneğin; aşağıda 5 elemanlı bir tamsayılar dizisi tanımlanmıştır:

```
int[] sayilar = new int[5];
```

- Dizilere veri atamak için, aşağıdaki söz dizimi kullanılır:

```
sayilar[indeks] = 1;
```

indeks: Dizideki verilere verilen sıra numarasıdır. C#'ta indeksler sıfırdan başlar.

- Dizilerden değer okumak içinse, aşağıdaki söz dizimi kullanılır:

```
int sayi = sayilar[indeks];
```

- Örneğin, 11 kişilik bir futbol takımındaki futbolcuların forma numaralarını ayrı ayrı değişkenlerde tutmak yerine, futbolTakimi adlı bir dizi içinde tutabiliriz.

```
short[] futbolTakimi = new short[11];
```

- Bu ifadeyle, futbolTakimi isminde, içinde 10 tane short değeri tutabilecek bir dizi tanımlanır. Şimdi, forma numaralarını diziye aktaralım:

```
futbolTakimi[0] = 1;
futbolTakimi[1] = 4;
futbolTakimi[2] = 5;
futbolTakimi[3] = 2;
futbolTakimi[4] = 8;
futbolTakimi[5] = 12;
futbolTakimi[6] = 19;
futbolTakimi[7] = 99;
futbolTakimi[8] = 22;
futbolTakimi[9] = 10;
futbolTakimi[10] = 11;
```

- Bütün değerlere tek tek erişmek, eğer bu değerler birkaç tane ise pek sorun olmayacaktır. Fakat çok sayıda değer içeren bir dizideki değerlerin tümü listelenmek istenirse, bu yöntem pek uygun olmayacaktır. Bunun için **foreach** döngüsü kullanılabilir. Şimdi, foreach döngüsü ile tüm elemanları bir **Console.WriteLine()** ile ekrana yazdıralım:

```
foreach (short futbolcu in futbolTakimi)
{
    Console.WriteLine(futbolcu);
}
```


- Bir diziyi tanımlarken, aynı zamanda elemanlarına ilk değerleri de verebilirsiniz:

```
tip[] diziismi = { deger1, deger2, deger3....degerN };
```

- Bu şekilde, **{** işareti içinde dizinin bütün değerleri girilir ve dizinin boyutu da *N* olur. Örneğin, önceki futbolTakimi dizisinin değerleri aşağıda görüldüğü gibi belirtilebilir:

```
short[] futbolTakimi = {1, 4, 5, 2, 8, 12, 19, 99, 22, 10, 11};
```


Bazı Dizi Özellikleri ve Metotları

- Diziler, C#'ın kullandığı .NET Framework'ten (C# hazır kütüphaneleri olarak düşünülebilir) gelen, yapısına özgü tanımlanmış özellikleri ve metotları barındırır. Bunlardan bazıları aşağıda açıklanmıştır.
- **Length:** Bu metot, dizideki toplam eleman sayısını döndürür.
- Aşağıda bununla ilgili iki örnek verilmiştir.

```
string[] isimler = new string[40];
Console.WriteLine(isimler.Length);
```

Çıktı: 40

```
short[] futbolTakimi = {1, 4, 5, 2, 8, 12, 19, 99, 22, 10, 11};
Console.WriteLine(futbolTakimi.Length);
```

Çıktı: 11

- **Clear:** Bu metot, içine parametreler alarak, dizinin belirtilen alanındaki değerleri temizler. İlk parametre, dizinin kendisidir. İkinci parametre, silme işleminin dizinin hangi indeks'inden başlayarak gerçekleştirileceğidir. Üçüncü parametre toplamda kaç eleman silineceğinin belirtildiği alandır. İkinci ve üçüncü parametreler tamsayı(int) olarak atanmalıdır.

```
int[] dizi = {50, 63, 64, 75};
  //1. elemandan itibaren siler
Array.Clear(dizi, 1, 2);

// Dizideki tüm elemanları siler
Array.Clear(dizi, 0, dizi.Length);
```


- **Reverse:** Bu metot, dizi elemanlarının sıralamasını indeks sırasına göre tersine çevirir.

```
string [] harfler = {"A","B","C"};
Array.Reverse(harfler);
Console.WriteLine(harfler[2]);
```

Çıktı: A

- **Sort:** Bu metot, dizi elemanlarını dizinin tipine bağlı olarak sıralar. Dizi metinsel ise alfabetik olarak, numerik ise rakamların büyüklüğüne göre sıralama yapılır.

```
string [] harfler = {"C","B","A"};
Array.Sort(harfler);
Console.WriteLine(harfler[2]);
```

Çıktı: C

- **IndexOf :** Bu metot, dizi içindeki bir elemanın indeksini döndürür.

```
decimal[] sonuclar = { 78, 99, 100, 12 };
decimal maksimum = 100;
Console.WriteLine(Array.IndexOf(sonuclar,maksimum).ToString());
```

Çıktı: 2

- Görüldüğü üzere 100 değerini tutan ***maksimum*** değişkeni, sonuclar dizisinin ***2.*** indisinde ***(3. sırada)*** olduğu için ***Array.IndexOf( )*** ’un sonucu konsola **2** olarak döndürülmüştür.

- **BinarySearch :** Bu metot, bir nesneyi bir dizi içinde arar, eğer bulursa bulduğu nesnenin indeksini tutar, bulamazsa negatif bir sayı tutar. BinarySearch'ü kullanabilmek için diziyi daha önce Sort ile sıralamalıyız.

```
string[] dizi={"ayşe","osman","ömer","yakup","meltem"};
Array.Sort(dizi);
Console.Write(Array.BinarySearch(dizi,"osman"));
```

- **Ancak** yalnızca tek boyutlu diziler Sort ile sıralanabilir. Dolayısıyla çok boyutlu dizilerde Sort ile sıralama yapılamaz ve de BinarySearch ile arama yapılamaz.

Örnek Uygulama-1

- Verilen bir string dizisini, ters sırada (sondan başa doğru) listeleyen C# programını yazınız.

```
string[] strDizi = { "Hasan", "Ali", "Osman", "Fatma", "Süreyya" };
int son = strDizi.Length - 1;
for (int i = son; i >= 0; --i)
{
    Console.WriteLine(strDizi[i]);
}
```


Örnek Uygulama-2

- `int dizi[] = { 5,6,7,8,78,45,0,30};` şeklinde verilen bir tamsayı dizisinin elemanlarının toplamını bulup ekrana yazan C# programını for ve foreach döngülerini kullanarak ayrı ayrı geliştiriniz.

```
int[] dizi = { 5, 6, 7, 8, 78, 45, 0, 30 };
int toplam = 0;
for (int i = 0; i < dizi.Length; ++i)
{
    toplam += dizi[i];
}
Console.WriteLine(toplam);
```

```
int[] dizi = { 5, 6, 7, 8, 78, 45, 0, 30 };
int toplam = 0;
foreach (int i in dizi)
{
    toplam += i;
}
Console.WriteLine(toplam);
```

Örnek Uygulama-3

- Tanımlanan sayılar dizisi içerisindeki sayıların negafit, pozitif ve işaretsiz olma durumlarını yanına yazdıran C# programını yazınız.

```
int[] sayilar = { 4, 5, -15, 22, -34, 3, 0, 7, 43, 100 };
int toplam = 0;
foreach (int sayi in sayilar)
{
    if (sayi > 0)
        Console.WriteLine(sayi + " Pozitif");
    else if (sayi < 0)
        Console.WriteLine(sayi + " Negatif");
    else
        Console.WriteLine(sayi + " İşaretsiz");
}
```


Örnek Uygulama-4

- Kullanıcıdan alınan metin içindeki sesli harf sayısını bulan C# console uygulamasını yazınız.

```
char[] harfler={'a','e','ı','i','o','ö','u','ü'};
string metin;
int sayac = 0;
Console.WriteLine("Metin giriniz : ");
metin = Console.ReadLine();
for (int i = 0; i < harfler.Length; i++)
{
    for (int j = 0; j < metin.Length; j++)
    {
        if (metin[j] == harfler[i])
        {
            sayac++;
        }
    }
}
Console.WriteLine(sayac);
```

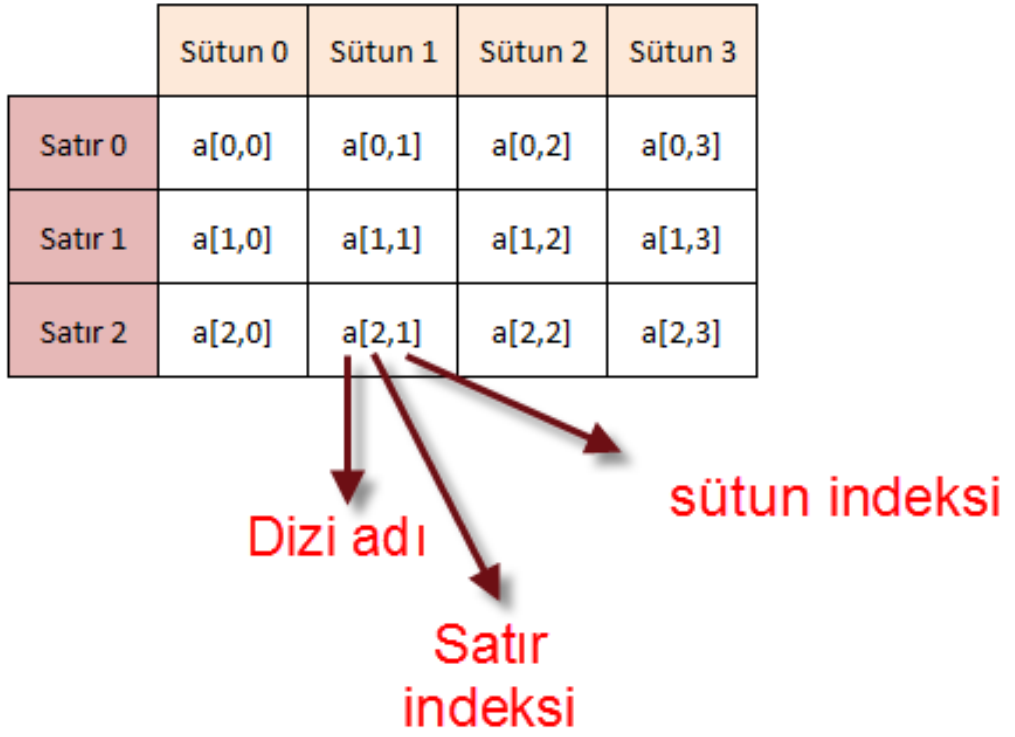
Örnek Uygulama-5

- 10 elemanlı bir dizide bulunan pozitif, sıfır ve negatif değerlerin sayısını bulup yazan programın C# programını yazınız.

```
int[] sayilar = { 10, -3, 5, 0, 33, -2, -60, 0, -20, -10 };
int sifirSay=0, pozitifSay=0,negatifSay = 0;
for (int x = 0; x < sayilar.Count(); x++)
{
    if (sayilar[x] < 0)
        negatifSay++;
    else if (sayilar[x] > 0)
        pozitifSay++;
    else
        sifirSay++;
}
Console.WriteLine("Sıfırların Sayısı : " + sifirSay);
Console.WriteLine("Negatiflerin Sayısı : " + negatifSay);
Console.WriteLine("Pozitiflerin Sayısı : " + pozitifSay);
```


Çok Boyutlu Diziler

- Çok boyutlu diziler daha çok satır ve sütunlar şeklinde oluşturulmuş veri kümelerindeki bilgileri saklamak için kullanılırlar.
- Örneğin iki boyutlu bir dizi tanımlama için dizi tanımlama esnasında hem satır hem de sütun bilgisini veririz. Burada dizi indisi tanımlama aşamasında ilk verilen değer satırı temsil eder, ikinci verilen değer sütunu temsil eder.
- Çok boyutlu dizi denilince sadece iki boyutlu diziler akla gelmemelidir. 2 boyuttan çok daha fazla boyutta diziler de oluşturulabilir. Her ne kadar pek kullanma ihtiyacımız olmasa da bu mümkündür.
- Diziler düzenli ve düzensiz olmak üzere iki gruba ayrılırlar. Yanda iki boyutlu düzenli bir dizinin şematik gösterimi verilmiştir.



- Yukarıda düzenli tipte verilen dizi, satırlar ve sütunlar şeklinde bilgi içeren tablo şeklinde verileri saklamak için kullanılır. Bu örnekte 3 satır ve 4 sütundan oluşan bir yapı görülmektedir.
- Bu yapıda dizimizin adı a'dır. a dizisinde tanımlanan ilk rakam yani ilk indeks satır indeksini verir. İkinci indeks ise sütun indeksini verir.
- İki boyutlu bir dizi aşağıdaki şekilde tanımlanır:

```
int[ , ] a = { { 10, 20 }, { 30, 40 } };
```

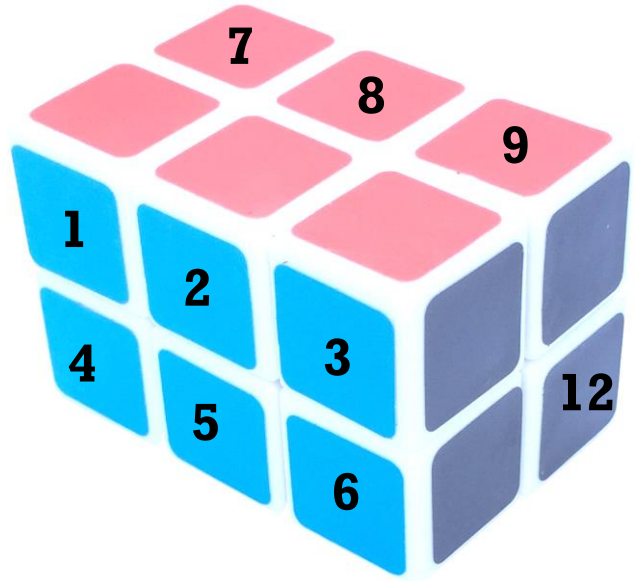
10	20
30	40

Tek boyutlu dizi tanımlamada olduğu gibi önce dizinin tipi sonra ise köşeli parantez kullanılır. İki boyutlu dizi olacağı için bir boşluk, bir virgül ve bir boşluk mevcut, yani virgül öncesi ve sonrası birer değer gelecek demek bu aslında. Dolayısıyla **iki boyutlu** dizileri oluşturmak için **[,]** ifadesi, **üç boyutlu** dizileri oluşturmak için ise **[, ,]** ifadesi kullanılır. Sonrasında ise eşitliğin sağ tarafında her bir satır küme parantezleri içine yazılır.

- Aşağıda ise örnek bir üç boyutlu dizi verilmiştir.

```
int[,,] dizi={ {{1,2,3},{4,5,6}} , {{7,8,9},{10,11,12}} };
```

- Dizi elemanları arasında verilen boşluklar tamamen anlaşılmayı kolaylaştırması için verilmiştir, isteğe bağlıdır.
- Parantezlerdeki renklendirmeler ise parantezlerin eşleştirilmesini kolaylaştırma amacı taşımaktadır.



- **GetLength :** Bu metot, dizinin boyutunu öğrenmek için kullanılır. Metotla beraber verdiğimiz indekse karşılık gelen dizinin uzunluğunu vermektedir. Aşağıdaki örneği inceleyelim.

```
int[,] dizi = { {12, 8, 9, 6}, {5, 3, 9, -4}, {4, 7, 8, -11}};
Console.WriteLine("Satır Sayısı: " + dizi.GetLength(0));
Console.WriteLine("Sütun Sayısı: " + dizi.GetLength(1));
```

- Bu iki boyutlu dizide **`dizi.GetLength(0)`** komutu satır sayısını, **`dizi.GetLength(1)`** komutu ise sütun sayısını döndürür. Programın ekran çıktısı aşağıdaki gibi olacaktır:

Satir Sayisi: 3

Sütun Sayisi: 4

Örnek Uygulama-1

- Üç boyutlu bir dizi tanımlayarak bu dizinin tüm elemanlarını alt alta ekrana yazdıran kodu geliştirelim.

```
int[,,] dizi = { {{1,2},{3,4}}, {{5,6},{7,8}}, {{9,10}, {11,12}} };
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 2; j++)
    {
        for (int k = 0; k < 2; k++)
        {
            Console.WriteLine(dizi[i, j, k]);
        }
    }
}
Console.ReadLine();
```


Örnek Uygulama-2

- Fibonacci dizisinin ilk 20 elemanını dizi yardımıyla hesaplayan ve ekrana yazdıran C# programını geliştiriniz.

```
int[] dizi = new int[20];
dizi[0] = 1; //ilk eleman
dizi[1] = 1; //ikinci eleman
for (int i = 2; i < dizi.Length; i++)
{
    dizi[i] = dizi[i - 1] + dizi[i - 2];
}
for (int i = 0; i < dizi.Length; i++)
{
    Console.WriteLine(dizi[i]);
}
```

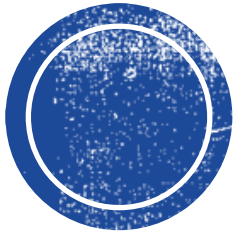

Örnek Uygulama-3

- Verilen iki boyutlu bir dizinin en küçük elemanını bulan C# programını geliştiriniz.

```
int[,] dizi = { {12, 8, 9, 6}, {5, 3, 9, -4}, {4, 7, 8, -11}};
int min = dizi[0,0];
for (int i = 0; i < dizi.GetLength(0); i++)
{
    for (int j = 0; j < dizi.GetLength(1); j++)
    {
        if (dizi[i,j] < min)
        {
            min = dizi[i, j];
        }
    }
}
Console.WriteLine("Minimum eleman: " + min);
Console.ReadLine();
```


Çalışma Soruları

- İki boyutlu bir dizinin tüm elemanlarını konsola yazdıran C# programını geliştiriniz.
- İki boyutlu bir dizinin tüm elemanlarınının toplamını hesaplayıp konsola yazdıran C# programını “foreach” döngüsü kullanarak geliştiriniz.
- İki boyutlu bir dizide kullanıcı tarafından istenen bir satırdaki en büyük elemanı bulan ve ekrana yazdıran C# programını geliştiriniz.
- İki boyutlu bir dizide kullanıcı tarafından istenen bir sütundaki elemanların aritmetik ortalamasını hesaplayıp ekrana yazdıran C# programını geliştiriniz.
- İki boyutlu bir dizinin tüm elemanlarının değerini iki katına çıkartan ve sonra bu diziyi ekrana yazdıran C# programını geliştiriniz.
- Kullanıcıdan alınan sayı uzunluğunda Fibonacci dizisi oluşturan ve ekrana yazdıran C# programını geliştiriniz.



METOTLAR

Bu bölümde C# programlama dilinde yeni metotların nasıl yazılabileceği Console uygulamaları üzerinden anlatılacaktır.

Metotlar

- Program yazarken, uygulamanıza iş yaptırabilmek için kod blokları oluşturursunuz. Bu kod blokları birden fazla yerde kullanılmak istendiğinde, aynı kodları tekrar yazmak yerine, yapılan işlemi bir metot olarak hazırlayabilir ve ihtiyaç duyduğunuz yerde metot ismi ile çağırarak çalıştırabilirsiniz. Bu size daha az kod yazma imkanı sağlar.
- Ayrıca, tek bir noktadan metodun çağrıldığı her yerdeki çalışma mantığında değişiklikler yapabilirsiniz. Bunun yanı sıra, kod tekrarı durumunda ortaya çıkan bir hatayı birçok yerde değiştirmek zorunda kalmaktan da metotları kullanarak kurtulabilirsiniz.

- Genel olarak metotlar, *geriye değer döndüren* ve *geriye değer döndürmeyen* metotlar olmak üzere ikiye ayrılır.
- *Geriye değer döndürmeyen metotlar*, verilen işi gerçekleştirip başka herhangi bir sorumluluğu olmayan bloklar olarak düşünülebilir.
- *Geriye değer döndüren metotlar* ise, oluşturulmuş iş blokları içindeki işlemler tamamlandıktan sonra geriye int, string vb. tiplerde değer döndürür. Döndürülen bu ifadeyi programınız içerisinde kullanmaya devam edebilirsiniz. Örnek olarak, Random sınıfını ele alabiliriz. Random, rastgele değer üretmeye yarayan bir sınıftır. Bu sınıfın Next adlı bir metodu daha vardır. Bu metodu çağırdığınızda size bir sonuç üretir ve siz onu programınızda uygun yerlerde kullanırsınız. Bu konuyla ilgi olarak aşağıdaki örneği inceleyelim:

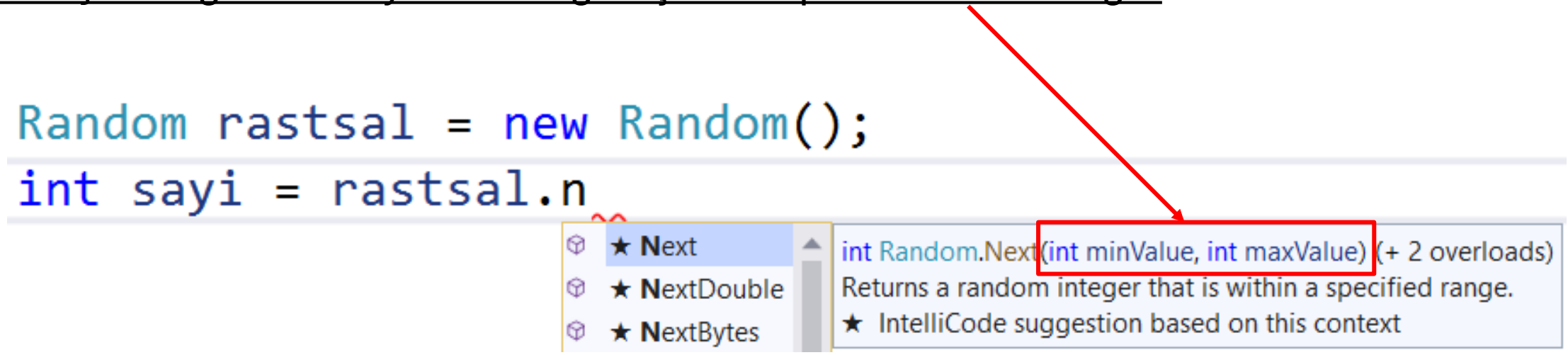
```
Random rastsal = new Random();
int sayi = rastsal.Next(1,5);
Console.WriteLine(sayi);
```

```
Random rastsal = new Random();
int sayi = rastsal.Next(1,5);
Console.WriteLine(sayi);
```

- Yukarıdaki örnek kod, sınırlar dahil [1,4] aralığında rastgele bir sayı oluşturarak bunu ekrana yazar.
- Görüldüğü üzere **Random** sınıfına ait **Next** metodu çağrılarak bir rastsal sayı üretilmiş ve bu değer **sayi** değişkenine aktarılmıştır. Daha sonra bu değişkenin değeri programda kullanılarak konsola yazdırılmıştır. Dolayısıyla burada kullanılan **Next** metodu değer döndüren metodlara bir örnektir.

Metotlarda Kullanılan Parametreler

- **Parametreler**, metotların istenen işlemi yapabilmesi için kullanıcı tarafından değerleri verilmesi gereken değişkenlerdir. Bir metotta istenen parametrelerin sayısına ve bu parametrelerin veri tiplerine o *metodun imzası* denir.
- Örneğin, **Random** sınıfının **Next** metodu istenen aralıkta bir rastsal sayı döndürmek üzere, en düşük değer ve en yüksek değer için **int** tipinde iki adet değer bekler.



- Bu değerleri 1 ve 101 olarak vermek için gereken kod aşağıdaki gibi yazılabilir:

sayi = rastgele.Next(1,101);

Kendi Metodunu Yazmak

- Bir metot şu şekilde tanımlanır:

```
tip metotismi(parametrelistesi)
{
    //yapılacak işler;
}
```

tip*:** Metodun geriye döndüreceği değerin tipidir. Metot geriye değer döndürmüyorsa buraya ***void yazılır.

***metotismi*:** Metoda verilen isimdir. Metot yazıldıktan sonra bu isimle çağrılacaktır.

***parametrelistesi*:** Gerekli hesaplamaların yapılması için metot içine dışarıdan verilebilecek değerlerdir.

- Bir metot, geriye değer döndürme ve parametre alma durumuna göre dört farklı şekilde tanımlanabilir:

ad() parametre almayan geriye değer döndürmeyen **void metod1()**	***dönüş tipi* ad()** parametre almayan geriye değer döndüren **double metod2()**
ad(*parametre listesi*) parametre alan geriye değer döndürmeyen **void metod3(int a, int b)**	***dönüş tipi* ad(*parametre listesi*)** parametre alan geriye değer döndüren **int metod4(int a, int b)**

- İlk olarak, verilen bir sayının karesini alan, yani o sayıyı kendisi ile çarpan bir metot oluşturalım. Metodun adı KareAl, geriye döndüreceği veri tipi **int** ve alacağı parametre veri tipi de **int** olsun:

```
int KareAl(int sayi)
{
    int sonuc;
    sonuc=sayi*sayi;
    return sonuc;
}
```

- Örnekte görüldüğü gibi, önce fonksiyonun geriye döndüreceği veri tipi belirtilir (**int**). Daha sonra, metoda isim verilir (KareAl). Parantezler içinde, metodun içine gönderilecek parametrenin veri tipi ve ismi belirtilir (int sayi).

- Metot içinde gerekli işlemler yapıldıktan sonra, geriye değer döndürebilmek için **return** ifadesi kullanılır. Eğer **return** ifadesi yazılmazsa veya metodun içindeki akış mantığı, bu ifadenin her durumda çalışmayacağı bir şekilde yazılırsa (örneğin bir **if** bloğu içine yazıldığında, koşulun gerçekleşip gerçekleşmeme durumuna bağlı olarak, **return** ifadesi çalışmayabilir) uygulama hata verir ve derlenmez. **return** ifadesi metodun çalışmasını durdurur ve sağındaki değeri geriye döndürür.
- Geriye değer döndürmeyen bir metot ise şu şekilde yazılabilir:

```
void Temizle()
{
    textBox1.Clear();
    textBox2.Clear();
    comboBox1.Items.Clear();
}
```


- Yukarıda kullanılan **Clear()** metodu, kullanıldığı kontrollerin içerisinde bulunan değerlerin temizlenmesi işlemini gerçekleştirir.
- İlk olarak, metodun bir değer döndürmediğini belirtmek için **void** metodu kullanılır. Sonra bu yeni metoda isim verilir, bkz. **Temizle**. Daha sonra, parantezler içine parametreler yazılır (bu örnekte metodun parametreye ihtiyacı yoktur).

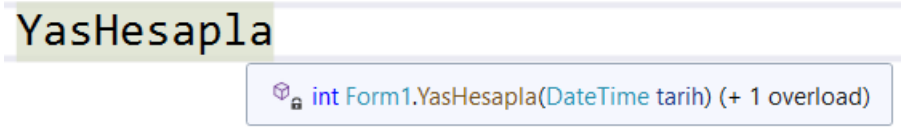
Metotları Aşırı Yükleme (Method Overload)

- Metotları aşırı yükleme (Method Overload), bir metodun farklı sürümlerinin hazırlanmasıdır. Bu sürümler, aynı ya da farklı amaçlar ile kullanılabilir ya da farklı tiplerde değerler döndürebilir. Ancak alınan parametreleri farklı olmalıdır.
- Overload operasyonunda aynı sayıda ve tipte parametre beklenen iki imza varsa, "Aynı imzaya sahip üye daha önce tanımlanmıştır." uyarısı alınır, dolayısıyla derleme hatası ortaya çıkar.
- Aşağıda verilen, aynı isme fakat farklı parametrelere sahip iki metodu inceleyelim.

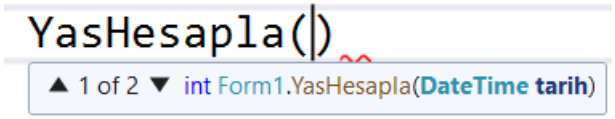
```
int YasHesapla(DateTime tarih)
{
    int sonuc;
    int dogumYili = tarih.Year;
    sonuc = DateTime.Now.Year - dogumYili;
    return sonuc;
}
```

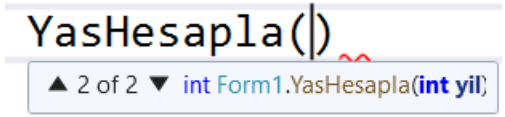
```
int YasHesapla(int yil)
{
    int sonuc;
    sonuc = DateTime.Now.Year - yil;

    return sonuc;
}
```


- YasHesapla ismini kullanarak ve imza gereksinimlerini karşıladıktan sonra, istediğiniz sürüme erişebilirsiniz.



- **DateTime** tipinde bir değer gönderirseniz ilk metot, **int** tipinde bir değer gönderirseniz ikinci metot çalışacaktır.
- Visual Studio'da bir metodun kaç overload'u olduğu, IntelliSense yardımıyla izlenebilir ve overload'lardaki imza farklılıkları buradan gözlemlenebilir.





Örnek Uygulama-1

- Kendisine gönderilen sayının faktöriyelini hesaplayıp geri döndüren metodun C# kodunu geliştiriniz.

```
private static double faktor(int alinansayi)
{
    double carpim = 1;
    for (int i = 1; i <= alinansayi; i++)
    {
        carpim *= i;
    }
    return carpim;
}
```


- Kendisine gönderilen iki boyutlu bir dizinin elemanları toplamını hesaplayıp geri döndürmeden sadece ekrana yazdıran metodun C# kodunu geliştiriniz.

```
private static void topla(int[,] alinandizi)
{
    double toplam = 0;
    foreach (int i in alinandizi)
    {
        toplam += i;
    }
    Console.WriteLine(toplam);
}
```

- Örneğin bu metot, main metodunun içerisinden aşağıdaki komutlar kullanılarak çağrılırsa ekrana yazırılacak değer 56 olacaktır.

```
int[,] dizi = { {12,8,9,6}, {5,3,9,-4}, {4,7,8,-11} };
topla(dizi);
```

Örnek Uygulama-2

- Kendisine gönderilen iki sayının *en büyük ortak bölenini* bulup ekrana yazdıran metodun C# kodunu geliştiriniz.

```
private static void gcd(int a, int b)
{
    int ebob = 1;
    int kucuk, buyuk;
    if (a<b){
        kucuk = a; buyuk = b;
    } else{
        kucuk = b; buyuk = a;
    }
    for (int i = 1; i <= buyuk; i++){
        if (kucuk % i == 0 && buyuk % i == 0){
            ebob = i;
        }
    }
    Console.WriteLine(ebob);
}
```

Çalışma Soruları

- Kendisine gönderilen tek boyutlu bir dizinin en büyük elemanını bulup ekrana yazdıran metodun C# kodunu geliştiriniz.
- Kendisine gönderilen iki boyutlu bir dizinin her bir satırının ayrı ayrı toplamını bulup ekrana yazdıran metodun C# kodunu geliştiriniz.
- Kendisine gönderilen iki boyutlu bir dizinin her bir satırının ayrı ayrı toplamını bulup ekrana yazdıran metodun C# kodunu geliştiriniz.
- Kendisine gönderilen bir sayının asal olup olmadığını bulan ve sonucu ekrana yazdıran metodun C# kodunu geliştiriniz.
- Kendisine gönderilen bir sayının asal bölenlerini bulup tek tek ekrana yazdıran metodun C# kodunu geliştiriniz.
- Kendisine gönderilen iki sayının *en küçük ortak katını* bulup ekrana yazdıran metodun C# kodunu geliştiriniz.