Bilgisayar Programlama II
Ders Notları

Öğr.Gör. Erkan Kaynak

İçindekiler

Bölüm 1 : Nesne Yönelimli Programlama (OOP) 3
OOP'nin Temel Kavramları 3
OOP'nin Avantajları 3
C# Dilinde Bir Sınıf Nasıl Yaratılır? 3
C# Dilinde bir Sınıftan Nesne Nasıl Yaratılır? 4
Sınıfların Elemanları (Members) Nelerdir? 4
Erişim Denetleyicisi Nedir ve Ne işe Yararlar? 5
İnşa Edici Metotlar (Constructors) Nedir? 5
C# Dilinde Kalıtım İşlemleri Nasıl Yapılır? 5
Bir Sınıfı Kalıtıma Kapatmak Mümkün mü? 5
Bölüm 2: Listeler 6
C# Dilinde Liste Nedir? 6
Ne İşe Yarar? 6
Liste Nasıl yaratılır ve Nasıl Eleman Eklenir? 6
Liste Elemanları ve Döngüler 6
Sık Kullanılan Liste Metotları ve Özellikleri 7
Daha Önce Kullandığımız Diziler ile Liste Arasındaki Farklar Nelerdir? 7
Bölüm 3: LINQ 8
Linq Kullanmanın Avantajları Nelerdir? 8
Linq ile Veri Filtreleme Nasıl Yapılır? 8
Linq ile Sıralama Nasıl Yapılır? 9
Sık Kullanılan Linq Metotları Nelerdir? 9
Bölüm 4: Tarih İşlemleri, Metinsel ve Sayısal İşlemler 11
C# Dilinde Tarih ve Zaman işlemlerini Nasıl Yaparız? 11
Şu anki Tarihi Almak 11
Belirli Bir Tarihi Oluşturmak 11
Tarih Bileşenlerine Erişmek 11
Tarih Formatlama 11
TimeSpan Yapısı Nedir? 12
TimeSpan ile Kendimiz Belirli Bir Süre Oluşturabilir miyiz? 12
Tarih Değerlerini Nasıl Karşılaştırırız? 12
Tarihe Gün, Saat, Dakika Ekleme 13
C# Dilinde En sık Kullanılan Matematiksel Fonksiyonlar Nelerdir? 13
C# Dilinde Sık Kullanılan Metinsel Fonksiyonlar Nelerdir? 14
Bölüm 5: Dosya ve Klasör İşlemleri 16
Klasör İşlemleri 16
Dosya İşlemleri 17

Bölüm 1 : Nesne Yönelimli Programlama (OOP)

OOP (Nesne Yönelimli Programlama / Object-Oriented Programming), yazılım geliştirmede kullanılan bir programlama paradigmasıdır. Bu yaklaşımda, yazılım nesneler (objects) adı verilen yapıların etrafında şekillenir. Her nesne, hem veri (özellikler / attributes / fields) hem de bu verilere yönelik davranışları (metotlar / functions) bir arada barındırır.

OOP'nin Temel Kavramları

1. **Sınıf (Class)**
   - Nesnelerin şablonudur. Bir nesnenin sahip olması gereken özellikleri ve metotları tanımlar.
   - Örnek: Araba sınıfı (renk, hız, çalıştır())
2. **Nesne (Object)**
   - Sınıftan türetilmiş gerçek bir varlıktır.
   - Örnek: araba1 = new Araba();
3. **Kalıım (Inheritance)**
   - Bir sınıfın, başka bir sınıfın özelliklerini ve metotlarını devralabilmesidir.
   - Örnek: SporAraba sınıfı, Araba sınıfından türeyebilir.
4. **Kapsülleme (Encapsulation)**
   - Verilerin doğrudan erişilmesini engellemek ve sadece kontrollü şekilde erişilmesini sağlamaktır.
   - Örnek: private değişkenler ve public getter/setter metotları.
5. **Çok Biçimlilik (Polymorphism)**
   - Aynı isimdeki metotların farklı şekillerde çalışabilmesidir.
   - Örnek: Araba sınıfındaki HareketEt() metodu, farklı alt sınıflarda farklı şekilde davranabilir.
6. **Soyutlama (Abstraction)**
   - Karmaşık sistemlerin sadece gerekli yönlerini gösterip, detayları gizlemektir.
   - Örnek: Kullanıcı sadece Çalıştır() metodunu çağırır ama motorun nasıl çalıştığını bilmek zorunda değildir.

OOP'nin Avantajları

- Kodun yeniden kullanılabilirliği
- Bakımın kolaylaşması
- Daha az hata ve daha yüksek güvenlik
- Gerçek dünya modellemesine uygunluk
- Büyük projelerin yönetimini kolaylaştırır

C# Dilinde Bir Sınıf Nasıl Yaratılır?

C# Dilinde bir sınıf yaratmak için Class anahtar kelimesi kullanılır. Araba isminde bir sınıf yaratalım ve renk adında bir değişkene sahip olsun:

```
class Araba
{
        public string Renk;
}
```

Bir sınıfın içerisinde bir değişken tanıttığımızda buna bir **Alan** (field, attribute) deriz.

C# Dilinde bir Sınıftan Nesne Nasıl Yaratılır?

Yukarıda bahsettiğimiz gibi, nesneler sınıflardan yaratılırlar. Araba sınıfımızdan yeni bir nesne yaratalım. Bunun için **new** anahtar kelimesi kullanılır:

```
Araba araba = new Araba();
araba.Renk = "Mavi";
```

Bir sınıfın Alanlarına (özelliklerine) erişmek için (.) notasyonu kullanılır. Nesne adından sonra nokta (.) yazılıp sonra ilgili özelliğe erişilebilir.

Bir sınıftan birden fazla nesne yaratılabilir.

```
Araba araba1 = new Araba();
Araba araba2 = new Araba();

araba1.Renk = "Beyaz";
araba2.Renk = "Siyah";
```

Sınıfların Elemanları (Members) Nelerdir?

C# Dilinde sınıfların Elemanları olarak ifade ettiğimiz 2 yapı vardır. Biri yukarıda gördüğümüz Alanlar (field) , diğeri ise Metotlardır.

Alanlar, bir sınıfın özelliklerini ifade eder. Renk, boy, hız vb. Her biri sınıfın altında bir değişken olarak tanımlanırlar.

Metotlar ise sınıfların yapabildiği eylemleri ifade ederler. BilgileriEkranaYaz, İlerle, Dur vb.

Örnek :

```
class Ogrenci {
        public string Adi;
        public string Soyadi;
        public int Numarasi;

        public void BilgileriEkranaYaz() {
                Console.WriteLine("Adı Soyadı: " + Adi +" " + Soyadi);
                Console.WriteLine("Numara: " + Numarasi);
        }
}

public static void Main() {
          Ogrenci ogr = new Ogrenci();
          ogr.Adi = "Mehmet";
          ogr.Soyadi = "Yılmaz";
          ogr.Numarasi = 2025001;
          ogr.BilgleriEkranaYaz();
}
```

Erişim Denetleyicisi Nedir ve Ne işe Yararlar?

Erişim Denetleyicileri (Access Modifiers) bir sınıfın ya da sınıf elemanının (alan veya metot) erişilebilme (görülebilme) seviyesini ayarlamaya yarayan kelimelerdir.

C# dilinde aşağıdaki erişim denetleyicileri kullanılır:

public	Kod tüm diğer sınıflardan erişilebilir.
private	Kod sadece sınıfın kendi içerisinden erişilebilir. Diğer sınıflara kapalıdır.
protected	Kod sadece sınıfın kendi içerisinden veya bu sınıftan türetilmiş başka sınıftan (bkz:kalıtım) erişilebilir.
internal	Kod sadece aynı projeden (assembly) erişilebilir.

İnşa Edici Metotlar (Constructors) Nedir?

İnşa edici metotlar nesneleri yaratırken kullanılan özel metotlardır. Nesneler yaratılırken bu metotlar otomatik olarak çalıştırılırlar. Nesnenin özelliklerine başlangıç değerleri atamak veya nesnenin ilk durumu ayarlamak için kullanılırlar.

```
class Araba
{
        public string Renk;

        public Araba() {
                Renk = "Mavi";
        }
}
```

İnşa edici metodun ismi sınıf ismi ile aynı olmalı ve herhangi bir geri dönüş tipi (int, string, void) yazılmamalıdır.

C# Dilinde Kalıtım İşlemleri Nasıl Yapılır?

Kalıtımın bir sınıftan başka sınıflar türetmek olduğunu yukarıda yazmıştık. Bu işlem ile Ana Sınıftaki (Parent) tüm özellik ve metotlar alt sınıfa (çocuk – child) aktarılır. C# dilinde bir sınıftan başka bir sınıf türetmek için iki nokta üst üste simgesi (**:**) kullanılır.

```
class Personel {
        public string AdiSoyadi;
        public decimal Maas;
}
class SatisTemsilcisi : Personel {
        public string Sube;
}
```

Bir Sınıfı Kalıtıma Kapatmak Mümkün mü?

C# Dilinde bir sınıftan başka alt sınıflar türetilmesin istiyorsan sınıfı yaratırken **sealed** anahtar kelimesi kullanılır. Aşağıdaki kod hata verecektir çünkü Personel sınıfından bir alt sınıf türetilemez.

```
sealed class Personel {  // ...   }
class Muhasebe : Personel { //...  }
```

Bölüm 2: Listeler

C# dilinde Liste (List), koleksiyonlar (collections) kategorisinde yer alan ve dinamik boyutlu, generic bir veri yapısıdır. System.Collections.Generic isim uzayında yer alır ve birçok öğeyle çalışmayı kolaylaştırır.

C# Dilinde Liste Nedir?

- List<T> sınıfı, aynı türdeki (T) verileri sıralı olarak tutan, eleman sayısı ihtiyaç oldukça artabilen veya azaltılabilen bir yapıdır.
- Dizi (Array) gibi çalışır ama boyutu dinamik olarak değiştirilebilir.
- Generic bir yapıdır: Yani tip bağımlıdır (List<int>, List<string>, List<Araba> gibi).

Ne İşe Yarar?

- Çok sayıda aynı türde veriyi saklamak için kullanılır.
- Eleman eklemek, silmek, aramak gibi işlemleri kolaylaştırır.
- Döngülerle kolayca erişilebilir.
- İçindeki öğeleri sıralayabilir, filtreleyebilir, dönüştürebilirsin.

Liste Nasıl yaratılır ve Nasıl Eleman Eklenir?

C# Dilinde yeni bir liste yaratmak için new anahtar kelimesi kullanılır. Hangi tipte liste olacağını belirtmek için sivri parantezler (< , >) kullanılır. Listeye tek bir eleman eklemek için Add komutu, başka bir koleksiyonu eklemek için de AddRange komutu kullanılır.

```
List<string> meyveler = new List<string>();
// Eleman ekleme
meyveler.Add("Elma");
meyveler.Add("Armut");
meyveler.Add("Muz");
string[] meyveSepeti = {“Karpuz”, “Üzüm”};
meyveler.AddRange(mevceSepeti);
```

Liste Elemanları ve Döngüler

Liste elemanlarına erişmek için köşeli parantezler ve indisler ([x]) kullanılır. Bir döngü ile tüm elemanlara erişmek istendiğinde, normal **for** döngüsü veya **foreach** döngüsü kullanılabilir.

```
// Elemanları yazdırma (1)
for (int i = 0; i < meyveler.length; i++)
        Console.WriteLine( meyveler[i] );
// Elemanları yazdırma (2)
foreach (string meyve in meyveler)
        Console.WriteLine(meyve);
```

Sık Kullanılan Liste Metotları ve Özellikleri

Metot	Açıklama
Add(item)	Listeye eleman ekler
Remove(item)	Belirtilen elemanı siler
RemoveAt(index)	Belirtilen indisteki elemanı siler
Insert(index, item)	Belirtilen konuma eleman ekler
Clear()	Tüm elemanları siler
Contains(item)	Eleman var mı diye kontrol eder
Count	Eleman sayısını döner
Sort()	Sıralama yapar
Reverse()	Elemanların sırasını ters çevirir

Daha Önce Kullandığımız Diziler ile Liste Arasındaki Farklar Nelerdir?

Özellik	Dizi (Array)	Liste (List<T>)
Boyut	Sabit	Dinamik
Eleman Ekleme	Mümkün değil (yeniden tanımlama gerekir)	Add() ile kolayca yapılır
Fonksiyonellik	Sınırlı	Geniş fonksiyon yelpazesi
Tip Güvenliği	Var	Var (generic yapı)

Bölüm 3: LINQ

Language Integrated Query 'nin kısaltması olan Linq, C# dili ile koleksiyonlar üzerinde kolayca sorgulamalar yapmamızı sağlayan bir kütüphanedir. Sayı metin gibi temel tiplerin dışında karmaşık nesnelerden oluşan listelerde bile kolayca sorgu, hesaplama ve gruplama gibi işlemleri yapabiliriz.

Linq Kullanmanın Avantajları Nelerdir?

- **Kolay Okunabilirlik ve Yazılabilirlik:** LINQ, SQL benzeri bir söz dizimi kullanarak kodun daha anlaşılır ve okunabilir olmasını sağlar.
- **Tip Güvenliği:** Derleme zamanında hata kontrolü yaparak güvenli kod yazılmasını sağlar.
- **Daha Az Kod Yazımı:** Geleneksel döngüler ve koşullar yerine daha kısa ve etkili sorgular yazmanıza olanak tanır.
- **Performans Artışı:** LINQ, optimize edilmiş sorgular sayesinde veri işlemlerini daha hızlı gerçekleştirebilir.
- **Veri Kaynağı Bağımsızlığı:** Aynı sorgu yapısını kullanarak farklı veri kaynakları üzerinde işlem yapabilirsiniz (veritabanı, koleksiyonlar, XML, JSON vb.).
- **Bakım Kolaylığı:** LINQ ile yazılmış kodlar daha modüler ve düzenli olduğu için bakım ve geliştirme süreçleri daha kolaydır.

Linq ile Veri Filtreleme Nasıl Yapılır?

LINQ ile veri filtreleme, belirli koşullara uyan verileri seçmek için kullanılır. Bunun için **Where** yöntemi kullanılır

```
int[] numbers = { 10, 20, 30, 40, 50, 60 };
// 30'dan büyük olanları filtrele
var filteredNumbers = numbers.Where(x => x > 30);
foreach (var num in filteredNumbers)
    Console.WriteLine(num);
```

Nesne listeleri ile de sorgulama yapılabilir. Örnek:

```
class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

```
List<Person> people = new List<Person>
{
    new Person { Name = "Ali", Age = 25 },
    new Person { Name = "Ayşe", Age = 30 },
    new Person { Name = "Mehmet", Age = 35 }
};
// 30 yaşından büyük olanları filtrele
var adults = people.Where(x => x.Age > 30);

foreach (var person in adults)
    Console.WriteLine(person.Name);
```

Linq ile Sıralama Nasıl Yapılır?

LINQ ile sıralama, verileri belirli bir kritere göre sıralamak için kullanılır. Bunun için kullanılan metotlar şunladır:

- OrderBy – Artan Sıralama,
- OrderByDescending – Azalan Sıralama,
- ThenBy – Sonraki Sıralama (artan)
- ThenByDescending – Sonraki Sıralama (azalan()

Örnek:

```
int[] numbers = { 50, 10, 40, 20, 30 };
// Küçükten büyüğe sıralama
var sortedNumbers = numbers.OrderBy(n => n);
foreach (var num in sortedNumbers)
Console.WriteLine(num);
```

Nesne listeleri ile de sıralama yapılabilir. Örnek:

```
class Person
{
   public string Name { get; set; }
   public int Age { get; set; }
}
```

```
List<Person> people = new List<Person>
{
   new Person { Name = "Ali", Age = 25 },
   new Person { Name = "Ayşe", Age = 30 },
   new Person { Name = "Mehmet", Age = 35 }
};

// Yaşa göre büyükten küçüğe doğru sırala
var adults = people.OrderByDescending(x => x.Age);

foreach (var person in adults)
   Console.WriteLine(person.Name);
```

Sık Kullanılan Linq Metotları Nelerdir?

- Eleman Seçme Metotları
  - First(): İlk öğeyi getirir.
  - FirstOrDefault(): İlk öğeyi getirir, yoksa varsayılan değeri döndürür.
  - Last(): Son öğeyi getirir.
  - ElementAt(): Belirtilen indeksdeki öğeyi getirir.
- Sayısal İşlemler
  - Count(): Eleman sayısını döndürür.
  - Sum(): Sayısal değerlerin toplamını hesaplar.
  - Average(): Sayısal değerlerin ortalamasını hesaplar.
  - Max(): En büyük değeri döndürür.
  - Min(): En küçük değeri döndürür.
- Veri Dönüştürme Metotları
  - ToList(): Veriyi listeye dönüştürür.
  - ToArray(): Veriyi diziye dönüştürür.
  - ToDictionary(): Veriyi sözlüğe dönüştürür.

Örnek:

```
using System;
using System.Linq;

class Program
{
   static void Main()
   {
      int[] numbers = { 10, 20, 30, 40, 50 };

      // Sayısal işlemler
      int totalSum = numbers.Sum();                // Toplam
      double average = numbers.Average();          // Ortalama
      int maxNumber = numbers.Max();               // En büyük sayı
      int minNumber = numbers.Min();               // En küçük sayı
      int count = numbers.Count();                 // Eleman sayısı

      // Sonuçları ekrana yazdırma
      Console.WriteLine("Toplam: " + totalSum);
      Console.WriteLine("Ortalama: " + average);
      Console.WriteLine("En büyük sayı: " + maxNumber);
      Console.WriteLine("En küçük sayı: " +  minNumber);
      Console.WriteLine("Eleman sayısı: " +count);
   }
}
```

Bölüm 4: Tarih İşlemleri, Metinsel ve Sayısal İşlemler

C# Dilinde Tarih ve Zaman işlemlerini Nasıl Yaparız?

C# dilinde tarih ve zaman işleri için DateTime ve TimeSpan yapıları kullanılır. Bunlar sayesinde Tarih ve Saat değişkenleri oluşturabilir, bu değerler arasında işlemler yapabilir, tarih ve saatler arasındaki farkları bulup bunları tarihsel birimlere (gün, dakika vb) dönüştürebiliriz.

Şu anki Tarihi Almak

```
DateTime bugun = DateTime.Today; // Yerel tarih
DateTime simdi = DateTime.Now; // Yerel tarih ve saat
```

Belirli Bir Tarihi Oluşturmak

```
DateTime customDate1 = new DateTime(2025, 6, 10, 15, 34, 0);       // 10 Haziran 2025, 15:34:00
Datetime customDate2 = DateTime.Parse("10.06.2025 15:34:00");   // 10 Haziran 2025, 15:34:00
```

Tarih Bileşenlerine Erişmek

```
DateTime customDate1 = new DateTime(2025, 6, 10, 15, 34, 0);          // 10 Haziran 2025, 15:34:00
Console.WriteLine("Yıl: " + now.Year);                                // 2025
Console.WriteLine("Ay: " + now.Month);                                // 6
Console.WriteLine("Gün: " + now.Day);                                 // 10
Console.WriteLine("Saat: " + now.Hour);                               // 15
Console.WriteLine("Dakika: " + now.Minute);                           // 34
Console.WriteLine("Saniye: " + now.Second);                           // 0
```

Tarih Formatlama

Standart tarih ve saat formatları için ToString() metoduna d,D,t veya T harflerinden birini yazmak yeterlidir.

```
DateTime tarih = new DateTime(2025, 6, 10, 15, 34, 0);              // 10 Haziran 2025, 15:34:00
Console.WriteLine(tarih.ToString("d"));  // 10.06.2025 (Kısa tarih formatı)
Console.WriteLine(tarih.ToString("D"));  // 10 Haziran 2025 Salı (Uzun tarih formatı)
Console.WriteLine(tarih.ToString("t"));  // 15:46 (Kısa saat formatı)
Console.WriteLine(tarih.ToString("T"));  // 15:46:30 (Uzun saat formatı)
```

Özel tarih ve saat formatları için yine ToString() metodu kullanılır. Bu kez istediğimiz tüm alanları tek tek belirtiriz. Aşağıdaki çeşitli örnekleri bulabilirsiniz.

```
DateTime tarih = new DateTime(2025, 6, 10, 15, 34, 0);                  // 10 Haziran 2025, 15:34:00
Console.WriteLine(tarih.ToString("dd/MM/yyyy"));      // 10/06/2025 (Gün/Ay/Yıl)
Console.WriteLine(tarih.ToString("MM-dd-yyyy"));      // 06-10-2025 (Ay-Gün-Yıl)
Console.WriteLine(tarih.ToString("yyyy.MM.dd HH:mm")); // 2025.06.10 15:46 (Tarih ve saat)
Console.WriteLine(tarih.ToString("dd MMM yyyy"));     // 10 Haz 2025 (Ay ismi ile)
Console.WriteLine(tarih.ToString("dddd, dd MMMM yyyy"));// Salı, 10 Haziran 2025 (Tam tarih)
Console.WriteLine(tarih.ToString("HH:mm:ss"));       // 15:46:30 (Sadece saat, 24 Saat formatı)
Console.WriteLine(tarih.ToString("hh:mm tt"));       // 03:46 PM (12 saat formatı)
```

TimeSpan Yapısı Nedir?

İki tarih değeri arasındaki farklı bulmak için kullandığımız yapıdır. İki tarih arasındaki fark hesaplanır ve bu sonuç bir TimeSpan değeri verir. Bu değeri farklı tarih birimlerine kolayca çevirebiliriz.

```
DateTime startDate = new DateTime(2025, 6, 1);
DateTime endDate = new DateTime(2025, 6, 10);
TimeSpan difference = endDate - startDate;                    // iki tarih arasındaki fark

Console.WriteLine("Toplam Gün: " + difference.TotalDays);     // iki tarih arasındaki gün sayısı
Console.WriteLine("Toplam Saat: " + difference.TotalHours);   // iki tarih arasındaki saat sayısı
Console.WriteLine("Toplam Dakika: " + difference.TotalMinutes); // iki tarih arasındaki dakika sayısı
```

TimeSpan ile Kendimiz Belirli Bir Süre Oluşturabilir miyiz?

Evet, TimeSpan ile iki tarih arasındaki farkı bulup saklayabildiğimiz gibi, kendimiz de istediğimiz kadar bir süreyi oluşturabiliriz.

```
TimeSpan sure = new TimeSpan(2, 30, 0); // 2 saat 30 dakika
Console.WriteLine(sure);
```

Tarih Değerlerini Nasıl Karşılaştırırız?

İki tarihi tıpkı sayısal değerleri karşılaştırdığımız gibi karşılaştırabiliriz. Bunun için <,>, <=, >= karşılaştırma operatörlerini kullanabiliriz.

```
DateTime tarih1 = new DateTime(2025, 6, 10);
DateTime tarih2 = new DateTime(2025, 6, 15);
if (tarih1 < tarih)    Console.WriteLine("tarih1, tarih2 'den önce.");
if (tarih1 > tarih)    Console.WriteLine("tarih1, tarih2 'den sonra.");
```

Tarihe Gün, Saat, Dakika Ekleme

Bir tarih değerine gün, saat gibi değerlerini ekleyip yeni tarih değerleri oluşturabiliriz. Bunun için AddDays(), AddHours() gibi ilgili komutlar kullanılır. Aşağıdaki örnekte bunları bulabilirsiniz.

```
DateTime customDate = new DateTime(2025,6,10 15:34:00);
// Farklı süreleri ekleyelim
 DateTime futureDate1 = customDate.AddYears(2); // 2 yıl ekleme
 DateTime futureDate2 = customDate.AddMonths(3); // 3 ay ekleme
 DateTime futureDate3 = customDate.AddDays(10); // 10 gün ekleme
DateTime futureDate4 = customDate.AddHours(5); // 5 saat ekleme
DateTime futureDate5 = customDate.AddMinutes(30); // 30 dakika ekleme
DateTime futureDate6 = customDate.AddSeconds(45); // 45 saniye ekleme

Console.WriteLine("Tarih " + customDate);
Console.WriteLine("2 yıl sonrası: " + futureDate1);
Console.WriteLine("3 ay sonrası: " + futureDate2);
Console.WriteLine("10 gün sonrası: " + futureDate3);
Console.WriteLine("5 saat sonrası: " + futureDate4);
Console.WriteLine("30 dakika sonrası: " + futureDate5);
Console.WriteLine("45 saniye sonrası: " + futureDate6);
```

C# Dilinde En sık Kullanılan Matematiksel Fonksiyonlar Nelerdir?

Mutlak değer

```
Console.WriteLine("Math.Abs(-5): " + Math.Abs(-5));                    // 5
```

Yuvarlama işlemleri

```
// Yukarı yuvarlama
Console.WriteLine("Math.Ceiling(4.3): " + Math.Ceiling(4.3));          // 5
// Aşağı yuvarlama
Console.WriteLine("Math.Floor(4.9): " + Math.Floor(4.9));                    // 4
// Yuvarlama - Varsayılan (En yakın tam sayıya)
Console.WriteLine("Math.Round(4.5): " + Math.Round(4.5));              // 4 (En yakın çift sayıya yuvarlar)
Console.WriteLine("Math.Round(4.6): " + Math.Round(4.6));              // 5
Console.WriteLine("Math.Round(4.567, 2): " + Math.Round(4.567, 2));    // 4.57
```

Büyük/Küçük değer

```
Console.WriteLine("Math.Max(10, 20): " + Math.Max(10, 20));            // 20
Console.WriteLine("Math.Min(10, 20): " + Math.Min(10, 20));            // 10
```

Kuvvet ve karekök

```
Console.WriteLine("Math.Pow(2, 3): " + Math.Pow(2, 3));                    // 8 (2^3)
Console.WriteLine("Math.Sqrt(16): " + Math.Sqrt(16));                      // 4 (√16)
```

Trigonometrik fonksiyonlar (Radyan cinsinden)

```
Console.WriteLine("Math.Sin(Math.PI / 2): " + Math.Sin(Math.PI / 2));      // 1 (Sin 90°)
Console.WriteLine("Math.Cos(0): " + Math.Cos(0)); // 1 (Cos 0°)
Console.WriteLine("Math.Tan(Math.PI / 4): " + Math.Tan(Math.PI / 4));    // 1 (Tan 45°)
```

Logaritma ve üstel fonksiyonlar

```
Console.WriteLine("Math.Log(10): " + Math.Log(10));                        // 2.302585 (Doğal logaritma)
Console.WriteLine("Math.Log10(100): " + Math.Log10(100));                  // 2 (10 tabanında logaritma)
Console.WriteLine("Math.Exp(1): " + Math.Exp(1));                          // 2.718281 (e^1)
```

C# Dilinde Sık Kullanılan Metinsel Fonksiyonlar Nelerdir?

Metnin uzunluğu

```
string text = "Merhaba Dünya!";
Console.WriteLine("text.Length: " + text.Length);                  // 14
```

Büyük/küçük harfe çevirme

```
string text = "Merhaba Dünya!";
Console.WriteLine("text.ToUpper(): " + text.ToUpper());            // MERHABA DÜNYA!
Console.WriteLine("text.ToLower(): " + text.ToLower());            // merhaba dünya!
```

Boşlukları temizleme

```
string textWithSpaces = "  Merhaba  ";
Console.WriteLine("textWithSpaces.Trim(): " + textWithSpaces.Trim());                  // "Merhaba"
Console.WriteLine("textWithSpaces.TrimStart(): " + textWithSpaces.TrimStart());       // "Merhaba  "
Console.WriteLine("textWithSpaces.TrimEnd(): " + textWithSpaces.TrimEnd());           // "  Merhaba"
```

Belirli bir bölümü alma

```
Console.WriteLine("text.Substring(8): " + text.Substring(8));                          // "Dünya!"
Console.WriteLine("text.Substring(8, 5): " + text.Substring(8, 5));                    // "Dünya"
```

Metin içinde değiştirme

```
Console.WriteLine("text.Replace(\"Dünya\", \"Evren\"): " + text.Replace("Dünya", "Evren"));        // "Merhaba Evren!"
```

Metni belirli bir ayraç ile bölme

```
string fruits = "Elma,Armut,Muz";
string[] fruitArray = fruits.Split(',');
Console.WriteLine("fruitArray[0]: " + fruitArray[0]);          // "Elma"
Console.WriteLine("fruitArray[1]: " + fruitArray[1]);          // "Armut"
Console.WriteLine("fruitArray[2]: " + fruitArray[2]);          // "Muz"
```

Metnin belirli bir karakter içerip içermediğini kontrol etme

```
Console.WriteLine("text.Contains(\"Dünya\"): " + text.Contains("Dünya"));          // True
```

Metnin belirli bir değerle başlayıp bitip bitmediğini kontrol etme

```
Console.WriteLine("text.StartsWith(\"Mer\"): " + text.StartsWith("Mer"));          // True
Console.WriteLine("text.EndsWith(\"!\"): " + text.EndsWith("!"));                  // True
```

Belirli bir karakterin indeksini bulma

```
Console.WriteLine("text.IndexOf(\"a\"): " + text.IndexOf("a"));                    // 4
Console.WriteLine("text.LastIndexOf(\"a\"): " + text.LastIndexOf("a"));            // 10
```

Metni belirli uzunlukta doldurma

```
Console.WriteLine("\"123\".PadLeft(6, '0'): " + "123".PadLeft(6, '0'));            // "000123"
Console.WriteLine("\"123\".PadRight(6, 'x'): " + "123".PadRight(6, 'x'));          // "123xxx"
```

Bölüm 5: Dosya ve Klasör İşlemleri

C# ile bilgisayarın dosya sistemine erişebilir, dosya ve klasör işlemleri yapabilirsiniz. Dosya ve klasör yaratma, silme gibi işletim sistemi komutlarını çalıştırabileceğiniz gibi dosya okuma ve yazma işlemleri ile depolama birimleri üzerinde de işlemler yapabilirsiniz.

Bunlar için System.IO uzayına referans vermeniz yeterli olacaktır. Sonraki işlemler için Directory ve File sınıflarını kullanacağız.

Klasör İşlemleri

- Directory.CreateDirectory(path) → Yeni bir klasör oluşturur.
- Directory.Delete(path, recursive: true/false) → Klasörü siler (recursive=true ise alt klasörleriyle birlikte).
- Directory.Exists(path) → Klasörün var olup olmadığını kontrol eder.
- Directory.GetFiles(path) → Klasördeki dosyaları listeler.
- Directory.GetDirectories(path) → Klasördeki alt klasörleri listeler.
- Directory.Move(sourcePath, destPath) → Klasörü taşır.
- Directory.GetParent(path) → Klasörün üst dizinini döndürür.
- Directory.GetCurrentDirectory() → Geçerli çalışma dizinini döndürür.

Örnek:

```
string folderPath = "C:\TestFolder";       // Kontrol edilecek klasör yolu

// Klasör var mı kontrol et
if (Directory.Exists(folderPath))
{
        Console.WriteLine("Klasör bulundu: " + folderPath);
        Console.WriteLine("İçindeki dosyalar:");

        // Dosyaları al ve ekranda listele
        string[] files = Directory.GetFiles(folderPath);
        foreach (string file in files)
        {
                Console.WriteLine(Path.GetFileName(file));          // Dosya adını yazdır
        }
}
else
{
        Console.WriteLine("Klasör bulunamadı: + folderPath");
}
```

Dosya İşlemleri

- File.Create(path) → Yeni bir dosya oluşturur.
- File.Delete(path) → Dosyayı siler.
- File.Exists(path) → Dosyanın var olup olmadığını kontrol eder.
- File.Copy(source, destination, overwrite: true/false) → Dosyayı başka bir konuma kopyalar.
- File.Move(source, destination) → Dosyayı taşır.
- File.ReadAllText(path) → Dosyanın içeriğini okur.
- File.WriteAllText(path, content) → Dosyaya metin yazar (eski içerik silinir).
- File.AppendAllText(path, content) → Dosyanın sonuna metin ekler.
- File.ReadAllLines(path) → Dosyayı satır satır okur (dizi olarak döndürür).
- File.WriteAllLines(path, lines) → Dosyaya satır satır yazma işlemi yapar.

Örnek :

```
string filePath = "ogrenci.txt";   // Dosya adı ve yolu

// Öğrenci bilgileri
string ad = "Ali Yılmaz";
int yas = 21;
string ogrenciNo = "2023001";

// Dosyaya yazılacak içerik
string icerik = "Öğrenci Bilgileri:" + Environment.NewLine +
                "------------------" + Environment.NewLine +
                "Ad: " + ad + Environment.NewLine +
                "Yaş: " + yas + Environment.NewLine +
                "Öğrenci Numarası: " + ogrenciNo;

// Dosyaya yazma işlemi
File.WriteAllText(filePath, icerik);

Console.WriteLine("Dosya oluşturuldu ve bilgiler yazıldı:“ + filePath);
```