

## İÇİNDEKİLER

<b>PROGRAMLAMANIN TARİHÇESİ .....</b>	<b>4</b>
<b>PROGRAMLAMAYA GİRİŞ .....</b>	<b>6</b>
Yazılım Nedir? .....	6
Kod Derleme Süreçleri (CLR, CLS, MSIL) .....	7
Microsoft Intermediate Language - MSIL (IL).....	8
CLR ( Common Language Runtime ) .....	8
CLS ( Common Language Specification - Ortak Dil Spesifikasyonu ) .....	9
<b>VISUAL STUDIO GIRIS .....</b>	<b>10</b>
Visual Studio Panelleri .....	12
Solution Explorer .....	12
Toolbox.....	12
Properties Window .....	13
Projeyi Derlemek ve Çalıştırmak.....	13
<b>PROGRAMLAMAYA BAŞLARKEN.....</b>	<b>13</b>
Yazılımcı Dostu Bilgiler.....	14
C#'da Yorum Satırı.....	14
Word Wrap .....	14
Line Numbers .....	15
Scope .....	15
Syntax.....	15
Indent Guides .....	16
Intellisense .....	16
Tooltip .....	16
Smart Tag.....	16
<b>YAZILIMA GİRİŞ .....</b>	<b>17</b>
Events (Olaylar).....	17
Event Mantığı.....	17
Event Detayları .....	18
Atama Operatörü .....	19
<b>VARIABLES (DEĞİŞKENLER).....</b>	<b>20</b>
Veri Tipleri.....	20
Sayısal Türler .....	20
Metinsel Türler .....	21
Mantıksal Türler .....	21
Object Veri Tipi.....	21
Var Kullanımı .....	22
Datetime (Tarih Zaman).....	22
Local ve Global Değişkenler .....	22
Local Değişkenler.....	22
Global Değişkenler .....	23
Constant (Sabitler) .....	24
<b>CONVERT İŞLEMLERİ .....</b>	<b>24</b>
<b>EXCEPTION HANDLING .....</b>	<b>24</b>
Syntax Errors (Derleme Zamanı Hataları) .....	24
Runtime Hataları (Çalışma Zamanı Hataları).....	24
Logical Hatalar – Mantıksal Hatalar .....	25
Hata Yönetimi.....	25

Error List.....	25
Try Catch Finally.....	25
Break Point .....	27
Open Exception Helper .....	27
<b>OPERATÖRLER.....</b>	<b>28</b>
Aritmetik Operatörler .....	28
Arttırma Operatörleri .....	28
Birleşik Operatörler.....	29
Mantıksal Operatörler.....	29
İlişkisel Operatörleri .....	30
Random Sınıfı .....	30
DateTime Sınıfı .....	30
Timer Kontrolü.....	31
<b>Karar Yapıları .....</b>	<b>31</b>
IF Else – Else IF .....	32
Tekmi Çiftmi Oyunu Uygulaması .....	33
Barbut Uygulaması .....	34
At Yarışı Uygulaması.....	36
Switch Case .....	37
Hangi Ay Uygulaması .....	38
Switch Case ile Ay'ın Hangi Mevsimde Olduğunu Bulma .....	39
Top Sektirme Uygulaması .....	40
Ternary IF .....	41
<b>Döngüler .....</b>	<b>42</b>
For döngüsünün yapısı.....	42
While Döngüsü .....	44
Do While Döngüsü .....	45
<b>Diziler .....</b>	<b>45</b>
Dizideki En Büyük ve En Küçük Elemanı Bul.....	46
Yazıyı Tersten Yazdırma.....	47
Foreach Döngüsü.....	47
Object Tipi .....	48
Boxing (Kutulama) .....	50
Unboxing (Kutudan Çıkarma).....	50
Tip Öğrenme .....	50
Var Tipi.....	50
<b>Metotlar.....</b>	<b>51</b>
Geriye Dönüş Tipi .....	51
Metot İsimlendirme.....	51
Geriye Değer Döndürmeyen ve Parametre Alan Metotlar .....	53
Geriye Değer Döndüren ve Parametre Almayan Metotlar .....	55
Geriye Değer Döndüren ve Parametre Alan Metotlar .....	56
Ref Parametresi.....	57
Out Parametresi ve Kullanımı .....	59
Params Kullanımı .....	60
Metot Overloading .....	61
Metot Summary .....	61
<b>String Metotlar .....</b>	<b>62</b>
Trim Metodu .....	62
Equals Metodu.....	62
SubString.....	63

Reverse .....	63
Index Of .....	63
Remove .....	63
Contains.....	64
Replace .....	64
Split.....	64
ToUpper .....	65
ToLower .....	65
Guid.....	65
2.Yöntem .....	66
Sadece Sayı, Metin, Özel Karakter Girişi .....	66
<b>Math Kütüphanesi .....</b>	<b>66</b>
Math.Pi .....	66
Math.Abs.....	67
Math.Ceiling .....	67
Math.Floor .....	67
Math.Max.....	67
Math.Min .....	67
Math.Pow .....	67
Math.Sqrt.....	68
Math.Round .....	68

## PROGRAMLAMANNIN TARİHÇESİ

Bilgisayar ve Programlama dünyasının gelişimine baktığınızda yalnızca 50 yıllık bir geçmişe sahiptir. Bu süreç içerisinde yaşanan değişim bugün bakıldığında inanılmaz boyutlardadır. Programlamanın tarihine bakmak ve gelişimini anlamak için öncelikle bilgisayarın gelişim sürecini incelemek gerekir.

Bilgisayarın gelişmesi, ona paralel olarak programlamanın da gelişmesi anlamını taşır. İnsanların sınırsız hayal gücünün ürünü olan bilgisayar, günümüzde insanların hayallerini gerçeğe çeviren bir icat olmuştur.

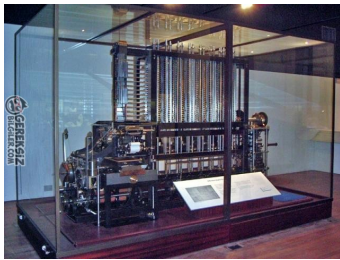
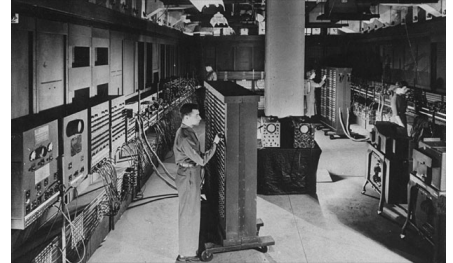
Günümüzde 1000'i aşkın programlama dili bulunmaktadır. Bilgisayarın gelişme sürecini incelerken dilleri isimleriyle değil, görevleriyle kullanacağız. Dilleri görevlerine göre aşağıdaki başlıklarda inceleyebiliriz.

- Bir amaç için tasarlanan programlama dilleri
- Geniş kitleler tarafından kullanılan ve programcılıkta kilometre taşı olmuş diller
- Kendinden sonraki dillere örnek olmuş dillerdir.

Dillerde insanlar gibi canlıdır, insanlar gibi doğar, bir gelişim sürecine girer ve güncel teknolojiye ayak uyduramadıkları gün ölürler. Programlamanın geliştiği 50 yıllık bu süreçte bir çok programlama dilinin doğumu ve ölümü gerçekleşti. Tabi diller geçirdiği evrimler, yakaladığı ve yarattığı yenilikler ile ömrünü uzatabilir. Her dil bir gün kendisini aşmak durumunda kalmalıdır, aksi takdirde yok olması kaçınılmazdır.

Bilgisayarın tarihine baktığımızda bu işi abaküsün icadına hatta mayaların yaptığı tarihsel ve matematiksel hesaplamaları yapmak için kullandıkları mekanizmalara kadar götürebiliriz. Fakat bizim bildiğimiz anlamıyla ilk bilgisayarlardan başlamak, yazının asıl amacından kopmamak adına daha faydalı olacaktır. Cambridge üniversitesinde 1822'de geliştirilmeye başlanan Differential Engine ve Analytical Engine adıyla iki makine tasarlanmış ve bu iki makine bugünkü bilgisayarların atalarından olmuştur. Fakat şunuda belirtmemiz gerekirk Differential ve Analytical Engine 1822 ve 1848 tarihleri arasındaki teknoloji yetersizliğinden dolayı asla çalışmamıştır.

Dünyadaki ilk bilgisayar **ENIAC** (*Electronic Numeric Integrator and Computer – Elektronik Sayısal Entegreli Hesaplayıcı*) adıyla duyurulmuştur. ENIAC 1941 yılında Amerika'nın II. Dünya savaşına katılmasıyla geliştirilmeye başlanmıştır. ABD'li bilim insanları tarafından geliştirilen projede ENIAC 167m<sup>2</sup> bir alan kaplamaktaydı ve 30 ton ağırlığındaydı. ENIAC projesi gizli olarak **Pennsylvania** Üniversitesine ait Elektronik bölümüne verildi. Projenin asıl amacı daha isabetli uzun top ve füze atışı hesaplamalarını hızlı ve hatasız bir şekilde yapmaktır.



Yaklaşık olarak \$500.000 maliyet ile ENIAC'ın denemelerine 1945 yılında başlandı. ENIAC 1947 yılında ise gerçek anlamda çalışabildi. 2 Eylül 1945'de Japonya'nın teslim olmasıyla birlikte savaş sona ermiş ve ENIAC'a gerek kalmamıştır. Savaş sonrası ENIAC hava tahminleri, atom enerji hesapları, rüzgar tüneli hesaplamaları gibi işlerde kullanılmıştır. 1951 yılında Endüstriyel amaçla kullanılmaya başlanmıştır. ENIAC'ın parçaları Amerikan Ulusal Müzesinde sergilenmektedir. ENIAC çok fazla güç tüketmekte ve yüksek ısı yaymaktaydı.

1949 yılında **Princeton** Üniversitesinde **EDVAC** (*Electronic Discrete Variable Automatic Computer – Elektronik Ayırık Değişkenli Otomatik Hesaplayıcı*) yaratıldı. EDVAC'ı ENIAC'dan ayıran en büyük özellik ondalık taban yerine ikilik tabanı kullanıyor olmasıdır. Aşağıdaki tabloda sayı sistemlerini ve açıklamalarını görebilirsiniz.

Taban	Açıklaması
<b>İkilik Sayı Sistemi</b>	İkilik (Binary) sayı sisteminde taban 2'dir. İkilik sayı sisteminde yalnızca 0 ve 1 rakamları kullanılmaktadır. Günümüzdeki bilgisayarlar ve hatta tüm elektrik/elektronik devreler ikilik sayı sistemini kullanmaktadır.
<b>Sekizlik Sayı Sistemi</b>	Sekizlik (Oktal) sayı sisteminde sadece 0-7 arası rakamlar kullanılmaktadır.
<b>Onluk Sayı Sistemi</b>	Onluk (Decimal) sayı sisteminde 0-9 arası rakamlar kullanılmaktadır. 10 adet sayı bulunduğundan bu sistemin tabanı 10'dur. Gündelik hayatımızda kullandığımız sayı sistemidir.
<b>Onaltılık Sayı Sistemi</b>	Onaltılık (Hexadecimal) sayı sisteminde 0-9 arası sayılar ve A-F arası harfler kullanılmaktadır. A=10, B=11, C=12, D=13, E=14, F=15 değerlerini temsil eder. Günümüzde Hexadecimal sayı sistemi makine kodlarını yazmak için kullanılır.

Yandaki tabloda Onluk sayı sistemiyle yazılan sayıların ikilik, sekizlik ve onaltılık sayı dönüşümleri ile ilgili örnekleri görebilirsiniz.

Binary Sayı sisteminde sadece 0 ve 1 rakamlarının kullanılarak Decimal sayı sisteminde verilmiş sayıların ikilik tabanda nasıl yazıldığını görüyoruz. Aynı şekilde onaltılık ve sekizlik sistemdede aynı şekilde dönüşümleri görebilirsiniz.

Sayı sistemleri arasında dönüşüm yapmak için çeşitli web siteleri ve programlar bulunmaktadır. Ayrıca dönüşüm işlemleri kolay olduğundan bir kağıt ve kalemle kolayca dönüşümleri yapabilirsiniz. Sayı sistemleri hakkında yeterli bilgi edindiğimize göre EDVAC'ı yakından incelemeye başlayabiliriz.

Decimal	Binary	Hexadecimal	Oktal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17
16	10000	10	20

EDVAC gerçek anlamda yaratılan ilk bilgisayar ünvanına sahiptir. Çünkü program ve verileri aynı anda bellekte saklayabilme özelliğine sahiptir. Bu özelliği sayesinde işlemleri istenilen sırayla kendisi yapabilmektedir. Bellekte bilgi tutan birim Binary Digit (bit)'dir. Bit ikilik sayı sistemiyle çalışmaktadır ve elektronik sinyal var/yok ifadeleriyle çalışır. Bit'ler bir araya gelip, Byte'ları oluşturmaktadır. Bellek dediğimiz birim byte'ların oluşturduğu manyetik durumlardır. Program dediğimiz yapı, bellekte bulunan byte'ların adreslerinin okunup veya yazılması demektir.

- Valfli makineler 1. Kuşak bilgisayarlardır.
- Transistör Bilgisayarlar 2. Kuşak bilgisayarlardır.
- Bütünleşik Devreli (Integrated Circuit Technology) 3. Kuşak bilgisayarlardır.
- Large Scale Integration 4. Kuşak bilgisayar olup, günümüzde kullanılmaktadır.

Günümüzdeki bilgisayarlarda veri ve programlar belleğe bit olarak yazdırılmaktadır. Bu bit dizelerine makine kodu denir. Makine kodlarını kullanarak bilgisayara bir veri yazdırmak çok zor bir iştir. Bu işlem çok zor olduğundan gelen sinyali algılayıp makine koduna çeviren yapılar oluşturulmuştur. Günümüzdeki işletim sistemlerinin atası da bu yapılardır. Bilgisayarın gelişimini inceledikten sonra artık programlamanın tarihine ve gelişimini inceleyebiliriz.

Programlamanın tarihine baktığımızda 1800'lere kadar gidebiliriz. Ancak bu programlama dilleri ile yapılanlar bilgisayar programları değildir. Bildiğimiz anlamda programlama dilleri ise bilgisayarın gelişmesiyle 1960'lı yıllarda gelişmeye başlamıştır. Doğal olarak bilgisayarın gelişmesiyle birlikte, bilgisayar üzerinde çalışacak olan programlarda gelişmektedir.

1940'lı yıllarda, bilgisayar gelişimini sürdürürken oluşturulan assembly dili ile program geliştirmeleri yapılmaktaydı. Assembly'nin zorluğundan dolayı doğal dile daha yakın olan programlama dilleri geliştirilmiştir. Alan Turing, John Von Neumann, John Mauchly, J. Presper Eckert ve Herman Goldstine tarafından geliştirilen ENIAC Coding System ilk programlama dillerinden biri olarak kabul görmektedir.

1951 yılında Grace Hopper ilk derleyiciyi tasarlamıştır. Daha sonrasında 1954 yılında FORTRAN (FORMula TRANslator) IBM 705 için tasarlanmıştır. FORTRAN'ın ilk derleyicisi ise 1957 yılında tasarlanmıştır. Fortran'da ki

bazı eksiklikleri gidermek için ALGOL (ALGOritmic Language) tasarlanmıştır ve bu dilin iki farklı uyarlaması bulunmaktadır. Hemen ardından 1959 yılında COBOL dili geliştirilmiş ve 1961 yılında ise ilk derleyicisi oluşturulmuştur.

1960'lara gelindiğinde 1962-1967 yılları arasında Simula isimli bir dil geliştirilmiştir. Geliştirilen Simula ilk nesneye yönelik programlama dilidir. Yine bu zamanlarda John George Kemendy ve Thomas Eugene 1964 yılında Basic (Beginner's All-purpose Symbolic Instruction Code) dilini geliştirmiştir. Niklaus Wirth tarafından 1968 yılında Pascal dili geliştirilmeye başlanmıştır ve 1970'de pascal ilk versiyonu ile yayınlanmıştır.

1970'li yıllara gelindiğinde günümüz programlama dillerinin tamamını etkileyen C dilinin temelleri atılmaya başlandı. 1972 yılında Dennis Ritchie tarafından Bell Laboratuvarlarında C dili geliştirilmiştir. Dennies Ritchie, Ken Thompson ile birlikte Unix işletim sistemi ve B programlama dili üzerinde çalışmıştır.

1979 yılında Bjarne Stroustrup tarafından C++ dili geliştirilmiş ve 1983 yılında C++ dili yayınlanmıştır. C++ dili, C dilini temel alarak geliştirilmiştir ve Nesneye yönelik programlama kavramını kabul etmektedir. 1986 yılında Bertrand Meyer tarafından geliştirilen Eiffel programlama dili C++ gibi nesneye-yönelik bir dildir.

Windows tabanlı uygulama geliştirmeyi sağlayan Visual Basic 1.0 Microsoft tarafından 1991 yılında piyasaya sürülmüştür. Sun Microsystem tarafından geliştirilmiş olan Java programlama dili 1995 yılında piyasaya 1.0 sürümüyle yayınlanmıştır. Dinamik web sayfaları oluşturulmasını sağlayan PHP Rasmus Lerdorf tarafından yine 1995 yılında tasarlanmıştır.

Gelişmenin son aşamasında ise Web Tabanlı programlama yaygınlaşmıştır ve bu sebeple php, asp, jsp gibi diller sürekli yenilenmekteydi. 2001 yılında microsoft bu gelişmeye ayak uydurarak .Net Framework geliştirme çatısını piyasaya sürmüş ve C#, Vb.Net gibi yenilikler getirmiştir.

Bu modül içerisinde .Net framework hakkında bilgiler verilmeye çalışılmıştır. C# programlama dilinin kullanımı ve programcılığın temel ilkelerine değinilmiştir.

## PROGRAMLAMAYA GİRİŞ

Daha önce programlamanın tarihine, bilgisayarın gelişim sürecine göz attık, bilgisayarın çalışma mantığı ve temel prensipleri üzerinde azda olsa bilgi sahibi olmuştuk. İlk bilgisayarlardan ve ilk dillerden bahsederek, günümüz bilgisayarları ve programlama dilleri ile ilişkilerini inceledik. Şimdi programlamaya girmeden önce bilmemiz gereken bazı terim ve bilgiler bulunmaktadır. Öncelikle yapacağımız işi ve kendimizi tanıyarak işe başlayalım.

### Yazılım Nedir?

Değişik ve çeşitli görevler yapma amaçlı tasarlanmış, elektronik aygıtların birbirleriyle haberleşebilmesini ve uyumunu sağlayarak görevlerini ya da kullanılabilirliklerini geliştirmeye yarayan makina komutlarıdır.

Yazılımcı/Programcı Nedir?

Çeşitli programlama dillerini kullanarak, insanların hayatını kolaylaştırmak, iş takibi yapmak ve kullanıcılara bilgi vermek amacıyla elektronik aygıtların çalıştırabileceği programları yazan kişiye yazılımcı/programcı denir.

Terimler ve Bilinmesi Gerekenler

Bu modül içerisinde ve diğer modüllerde karşınıza çıkabilecek, günlük bilgisayar kullanımı sırasında bile sıkça duyduğumuz bazı terimleri öğrenerek işe başlamalıyız. Bu terimleri öğrenmek programlamayı hem daha hızlı öğrenmenizi sağlayacak hemde eğitim içeriğine adapte olmasını sağlayacaktır. Aşağıdaki tabloda önemli terim ve açıklamaları bulabilirsiniz. Tabloyu incelemmeden önce aşağıdaki tanımlardan bazılarını zamanla modüller içerisinde detaylı olarak değinileceğini unutmayın.

Terim	Açıklaması
<b>Compiler</b>	Herhangi bir dilde yazılan kodları derleyerek ara dil'e dönüştüren derleyicidir. Yazılan kodların makine diline dönüşmesi için gerekli olan ara dile dönüştürülmesini sağlar.
<b>Compile</b>	Yazılan kodları derleyiciye göndererek derleme işlemi yapılmaktadır. Derleyicinin yaptığı derleme işlemine Compile ismi verilmektedir.

<b>Object Oriented Programming</b>	Nesneye yönelik programlama, günlük hayattan nesneleri bilgisayar ortamına tanımlamamızı sağlayan bir yöntemdir.
<b>Java Runtime Environment</b>	Java Runtime Environment (JRE) java programlama dilinin derleyicisidir. Java kodlarını çalıştırmak için bilgisayara yüklenmesi gerekir. C#'da olduğu gibi yazılan programları makine diline dönüştürülmesini sağlar.
<b>Common Language Runtime</b>	Common Language Runtime (CLR) C# programlama dilinin derleyicisidir.
<b>Framework</b>	Programlama dillerinin komutlarını tanıyan ve bu kodları işleyen kütüphanedir. .net Framework 2.0 dan başlamış ve şu anda son olarak 4.5 versiyonu kullanılmaktadır. Örneğin Windows işletim sisteminde, Java Runtime Environment yüklü olmadığında java dili ile yazılan bir programı çalıştıramayız. Aynı şekilde .net Framework yüklü olmayan bir bilgisayarda C# kodları çalıştırılmaz.
<b>Integrated Development Environment</b>	Bütünleşik Geliştirme Ortamı ( <i>Integrated Development Environment - IDE</i> ) Yazılım geliştiricilerinin hızlı ve rahat bir şekilde yazılım geliştirebilmesini amaçlayan, geliştirme sürecini organize edebilen birçok araç ile birlikte geliştirme sürecinin verimli kullanılmasına katkıda bulunan araçların tamamını içinde barındıran bir geliştirme ortamıdır.
<b>Visual Studio</b>	Microsoft tarafından geliştirilen ve bir çok programlama dilini destekleyen bir IDE'dir.
<b>C Sharp (C#)</b>	Microsoft tarafından 2001 yılında duyurulan, C++ dili üzerine geliştirme yapılarak .Net Framework içerisinde ki, Object Oriented tekniğini destekleyen bir yazılım geliştirme dilidir.
<b>.Net Framework</b>	Çatısı altında C# dahil bir çok programlama dili bulunan bir geliştirme kütüphanesidir.
<b>Syntax</b>	Sözdizimi anlamına gelmektedir. Bir programlama dilinde ki ifadelerin uyulması gereken kurallar bütünüdür.
<b>Intermediate Language</b>	Ortal Dil ( <i>Intermediate Language - IL</i> ) anlamına gelmektedir. Daha sonraki konularımızda detaylı olarak değineceğiz.

## Kod Derleme Süreçleri (CLR, CLS, MSIL)

Bir programlama dili öğrenirken, kullandığımız dili tanımak ve onu en etkin şekilde kullanmak esastır. Kullanacağımız .net Framework platformunu ve C# programlama dilini daha yakından tanımak için kod derleme süreçlerini çok iyi bilmeliyiz. Öncelikle işlemleri basitçe tanıyalım ve daha sonra terimlerimizi detaylı olarak inceleyelim. Örneğin C# programlama dilinde bir program yazdık. Bu programı çalıştırdığımızda neler oluyor? Hangi işlemler gerçekleşiyor? Bu işlemler gerçekleşirken .net Framework bu işin neresinde yer alıyor? Öncelikle .Net Framework'ü daha yakından tanıyalım.

### .NET Framework

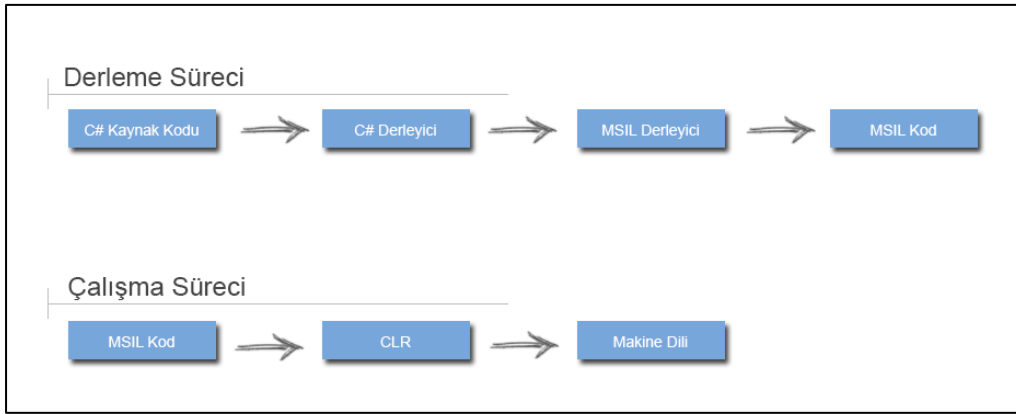
.Net Framework çatısı farklı dillerin aynı ortamda çalışmasını sağlar. Net Framework barındırdığı ortak dil çalışma zamanı (Common Language Runtime) sayesinde, .Net'in desteklediği diğer diller ile yazılmış kodları makine diline çevirerek yazılımlarda çoklu dil kullanmaya olanak sağlar. Çoklu dil kullanırken;

- Her kod önce kendi diline ait derleyici ile derlenir ve MSIL koduna dönüştürülür.
- Derlenen kod MSIL'e göre yeniden derlenir. Elde edilen MSIL kodu kendi başına çalışabilen bir kod değildir. MSIL bir takım taşıyabilir assembly koddan oluşur.
- Daha sonra CLR, MSIL olarak derlenmiş kodu çalıştırılabilir kod haline getirir.



Bu işlem tüm .Net Framework çatısı altında toplanan diller için aynı şekilde uygulandığı için .Net çatısı altında toplanan diller arasında performans farkı yok denilecek kadar azdır. Oluşan performans farkı sadece dile ait derleyiciden kaynaklanmaktadır. Net Framework çatısı altında aşağıdaki programlama dilleri bulunmaktadır.

- |                    |            |                           |
|--------------------|------------|---------------------------|
| ▪ C#               | ▪ Perl     | ▪ Oberon                  |
| ▪ Visual Basic.Net | ▪ Eiffel   | ▪ Salford FTN95 (Fortran) |
| ▪ Visual J#.Net    | ▪ Python   | ▪ Small Talk              |
| ▪ Visual C++       | ▪ Pascal   | ▪ Standart ML             |
| ▪ Jscript.Net      | ▪ Mercury  | ▪ Dyalog APL              |
| ▪ Cobol            | ▪ Mondrian |                           |



## Microsoft Intermediate Language - MSIL (IL)

Bir C# kodu yazıp derlediğimizde bu kod Microsoft Intermediate Language ( MSIL ) 'a dönüştürülür. Bu kod " sözde kod " ( pseudo code ) içeren bir dosyadır. Bu kod çalıştırılabilir bir kod değildir. Bu kod anca bulunduğu sistemde bir ara program ile çalıştırılır.

## CLR ( Common Language Runtime )

CLR kısaca yazılan programın tüm işletim sistemlerinde çalışmasını sağlamakla görevlidir. Örneğin biz java ile yazılmış bir oyunu Windows işletim sistemine sahip bir bilgisayarda oynamak istediğimizde, Java Runtime Environment'ı bilgisayarımıza kurmalıyız. CLR .NET ile programların çalışmasını kontrol eden birimdir. Eskiden programlar derlenirken makine diline çevrilirdi ve işletim sistemiyle bağlantılı olarak çalışırlardı. Ama günümüzde kodları yorumlayacak kütüphane ve derleyici programlar ile platform bağımsız kod yazabilmekteyiz.

Normalde Windows, Linux, MAC OS, Pardus aynı program kodunu çalıştıramazlar. Programların her biri için ayrı ayrı yazılıp derlenmesi gerekir. Bunun çözümü ise ortak bir ara dil kullanmak ve her bir platform için bu aradile çevrilmiş program kodunu çalıştırabilecek bir altyapı hazırlamaktır.

.NET de kütüphanelerle program kodu derlenip Microsoft Intermediate Language (Microsoft Ara dili) ' ne çevrilir. Burada oluşturulan Assembly CLR tarafından herhangi bir sistemde çalıştırılabilir.

Ortak dil çalışma platformu, .Net alt yapısında programların çalışmasını denetler ve programın işletim sistemi ile iletişim kurmasından sorumludur. C# program kodu MSIL Assembly'e, Assembly'i de CLR'e çevrilerek çalıştırılabilir bir makina koduna dönüşür. Bu .NET programlarının derlenip çalıştırılma mantığıdır.



C# ile üretilmiş kodların, MSIL koduna dönüştürüldükten sonra sistem üzerinde çalıştırılmasını sağlayan mekanizmadır. CLR'in görevi MSIL'i sistem üzerinde çalıştırılabilir koda dönüştürmektir. Yani CLR 'nin olduğu her ortamda ( işletim sistemi ve işlemci farketmeksizin ) C# da derlenmiş kodlar daha doğrusu MSIL çalışacaktır.

Program çalışacağı zaman C# ile derlenen kodlar yani MSIL kodlar; bir JIT ( Just-In-Time - Tam zamanında ) derleyici tarafından kullanılarak çalıştırılabilir koda dönüştürülürler. C# ( veya .NET çatısı altındaki herhangi bir dil ) kodları çalıştırıldığı zaman CLR, bu JIT derleyiciyi çalıştırır. JIT derleyici programınızın ihtiyaçlarına göre MSIL 'i yerel dile çevirir. Yani MSIL olarak ürettiğiniz her kod CLR 'nin olduğu her ortamda o sisteme uygun bir dile çevrilir ve yürütülür. Bu kodun içinde yer alan metadata olarak tanımlanan çıktı bulunur. Bu çıktı sayesinde programınızın diğer kodlarla etkileşimi sağlanır.

CLR ,programlarımızı değişik şekilde derleyebilir. Varsayılan derleme türü JIT(Just In Time- Çalışma Anında Derleme) 'dır. Program çalışırken daha önce derlenmemiş bir parçasına gelince hemen o kısmı da derler ve bunu hafızda chach'e koyar. Tekrar aynı program parçasını çalıştırmak gerekirse burayı hafızadan çalıştırır. Eğer RAM 'imizi yeteri kadar büyükse, programın tamamı derlenmiş ve hafızada depolanmış durumda olabilir. Bu durumda programımız çok hızlı çalışır.

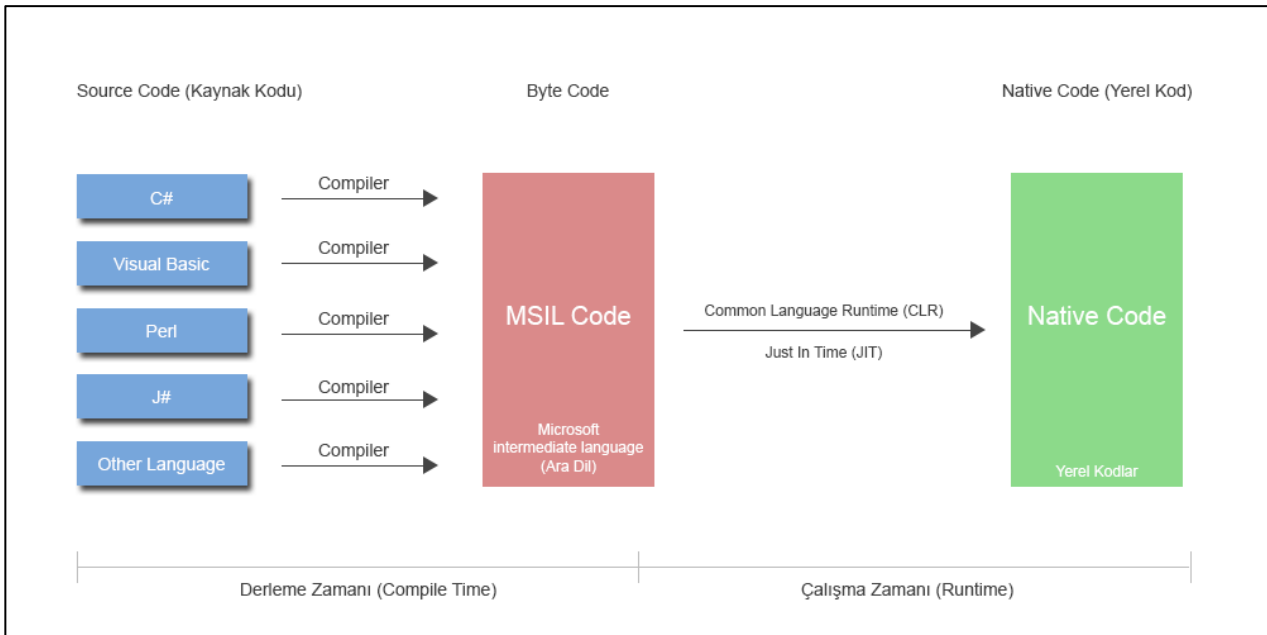
Hafızamızın yeteri kadar büyük olmadığı durumlarda EconoJIT (Ekonomik JIT) derleyicisini kullanabiliriz. Bu derleyici ile programın derlenmiş kısımları hafızada depolanmaz ve her seferinde aynı program parçası derlenir. Tabi ki bu derleyici normal JIT'e göre programlarımızı daha yavaş çalıştırır. Ama RAM 'imizi çok daha az kullanır.

CLR ile gelen üçüncü derleyicimiz PreJIT(ön JIT derleyicisi) ise derleme işini program çalışmadan önce yapar ve tüm makine kodlarını bir yerde saklar. Çalışma anında çok hızlı olan programımız diğer JIT derleyicileriyle derlenmiş olanlara nazaran çok hızlı çalışır. Kısaca C# kodumuz iki defa derleme aşamasından geçer program kodu MSIL'ye, MSIL ise makine koduna çevrilir.

## CLS ( Common Language Specification - Ortak Dil Spesifikasyonu )

Common Language Specification(CLS) bünyesinde barındırdığı birtakım yapıları ve kısıtları ile kütüphane(library) ve derleyici(compiler) yazabilmek için rehberlik yapmaktadır. CLS yazılan bir kütüphanenin CLS'yi destekleyen diğer programlama dilleri ile entegre şekilde çalışabilmesini ve bu diller tarafından da kullanılabilmesini sağlamaktadır. CLS, CTS'nin bir altkümesidir. CLS uygulama geliştiriciler için büyük önem arz etmektedir. Öyle ki bir uygulama geliştirici yazdığı kodun diğer kod geliştiriciler tarafından da kullanılabilir olmasını gözönünde bulundurmalıdır. CLS'nin kriterleri ve kuralları gözönünde bulundurularak yazılan bir API(Application Program Interface) diğer programlama dilleri içerisinde kullanılabilmekte Common Language Runtime tarafından da işletilebilmektedir.

.NET uyumlu programlama dili oluştururken belirli standartlara uyulması gerekir. Bu standartlar CLS (Common Language Specifications - Dillerin ortak özellikleri) ile belirlenmiştir.



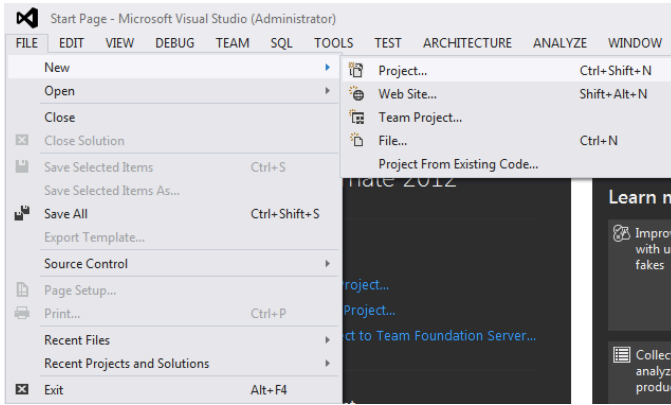
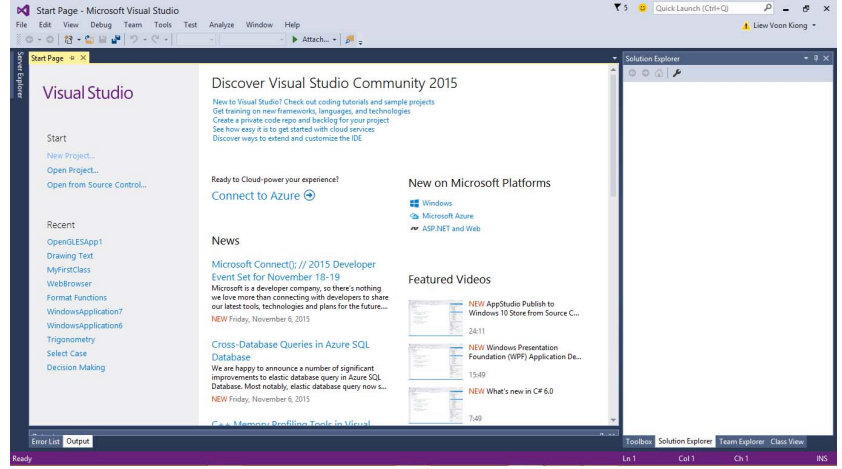
## VISUAL STUDIO GIRIS

Microsoft tarafından geliştirilen ve bir çok programlama dilini destekleyen bir IDE'dir. Birçok programlama dilini ve standartları destekler. Eklentileri ile geliştirici araçlarını artırır ve kod yazmayı kolaylaştırarak hızlandırır.

Bizler C Sharp kodlarımızı Visual Studio isimli IDE'de yazacağız. Yavaş yavaş Visual Studio'yu tanımaya başlayabiliriz.

Visual Studio açıldığında karşımıza Start Page ekranı gelecektir. (Resim 1.1)

Bu ekran Visual Studio kullanıcıları için bazı bilgilendirmeler ve kısa yollar sağlamak için bulunmaktadır.



**New Project** : Yeni bir proje oluşturmak için kullanabileceğimiz kısa yoldur. Dilersek; File menüsü altından New > Project yolunu izleyerek proje oluşturabiliriz.

**Open Project** : Daha önceden oluşturduğumuz bir projeyi açmak için kullanabileceğimiz seçenektir. Projeyi açmak için açılan pencereden proje yolu bulunur ve proje içerisindeki .sln uzantılı dosya seçilerek açılır.

**Solution Dosyası (.sln)** : Microsoft Visual Studio içerisinde program hazırlarken daha düzenli çalışmayı sağlayan metin tabanlı bir dosyadır. '.sln' uzantılı dosyaları çalıştırmak için Visual Studio programına ihtiyaç vardır.

**Recent** : En son kullandığımız projelerin bilgilerini tutar. Recent panelinin amacı, üzerinde çalıştığımız projeyi Microsoft Visual Studio programını çalıştırdığımızda daha kolay bir yöntem ile vakit kaybetmeden çalıştırmaktır. Burda projelere sağ tıklayarak listeden kaldırabilir, sln dosyasının bulunduğu klasörü görme işlemleri gibi özellikleri kullanabiliriz.

**Get Started** : Get Started paneli Microsoft Online'da paylaşılan Microsoft Visual Studio'yu tanıtmaya yönelik videoları içeren bir paneldir.

Microsoft Visual Studio'da Yeni Bir Proje Oluşturmak için aşağıdaki adımları takip ediniz.

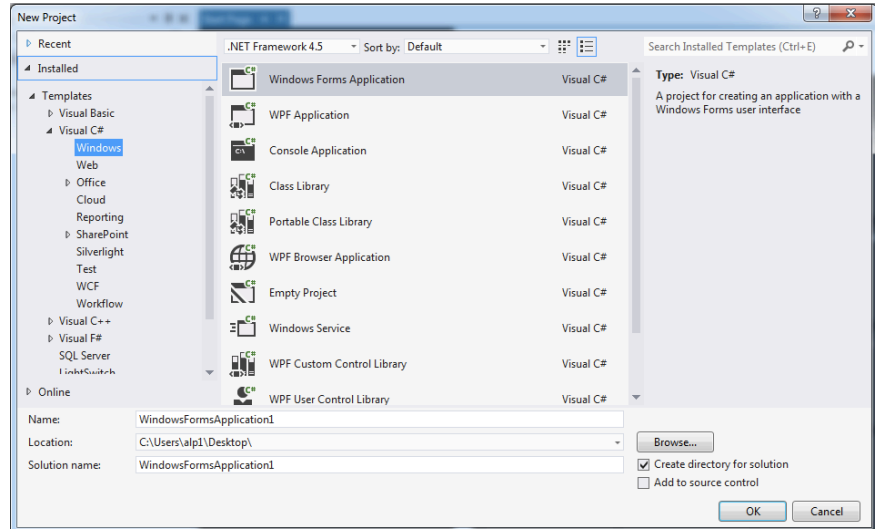
**Yöntem 1:**

File > New > Project yolunu takip ederek yeni bir proje oluşturma penceresini açabilirsiniz.

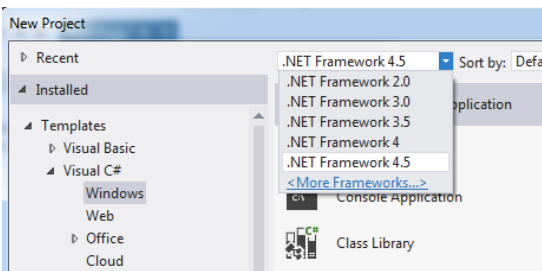
**Yöntem 2:**

Start Page sayfasında bulunan New Project bağlantısına tıklayarak yeni proje oluşturma penceresine ulaşabilirsiniz.

Açılan pencerede sol tarafta bulunan Templates seçeneği altından, Visual C# seçeneği seçilir. Visual C# altında bulunan kategorilerden hangi platform için çalışacağımızı seçmeliyiz. Platform ile ilgili kısa açıklamaları aşağıda bulabilirsiniz.



Template	Açıklama
<b>Windows</b>	Windows sekmesi altından seçtiğiniz programlama dili ile Desktop uygulamaları geliştirmek için kullanılan yapıları tutar.
<b>Web</b>	Web sekmesi altından seçtiğiniz programlama dili ile web uygulamaları geliştirmek için kullanılabileceğiniz yapıları tutar.
<b>Office</b>	Seçili programlama dili ile Microsoft Office programlarının özelliklerini kullanılabileceğimiz proje dosyalarını içerir.
<b>Cloud</b>	Microsoft Azure platformu üzerinde çalışacağımız zaman kullanacağımız proje dosyalarını içerir.
<b>Silverlight</b>	Silverlight uygulamaları geliştirirken kullanacağımız yapıları içerir.
<b>WCF</b>	Servis odaklı mimariye dayalı yapıları içerir. Projelerimizde servis çalıştırmak istediğimizde gerekli yapıları bu sekme altından oluşturabiliriz.



Açılan pencerede bulunan alandan projemizin .net Framework versiyonunu seçmemiz gerekmektedir. Framework versiyonları arasında ki farkdan ilerleyen konularda bahsedeceğiz.

Kullanacağımız template'i seçtikten sonra istediğimiz item'ı belirlememiz gerekiyor. Biz ilk olarak "Windows Form Application" nesnesini seçiyoruz.

.Net Framework versiyonunu seçtikten sonra hemen yanında bulunan Sort By seçeneği ile aşağıda bulunan platformları isimlerine göre sıralayabilirsiniz. Yeni proje oluşturma penceresi içerisinde aşağıdaki alanı görebilirsiniz.

Name:	WindowsFormsApplication1	
Location:	C:\Users\alp1\Desktop\	Browse...
Solution name:	WindowsFormsApplication1	<input checked="" type="checkbox"/> Create directory for solution <input type="checkbox"/> Add to source control
		OK Cancel

Projemiz ile ilgili isim, kayıt yeri ve solution bilgilerini belirleyebiliyoruz.

**Name** : Oluşturduğumuz projemizin ismidir.

**Location** : Oluşturulan proje dosyası hangi dizine kayıt edilecek bilgisidir.

**Solution** : İçerisinde birden fazla proje barındıran yapıdır.

**Create Directory for solution** : Proje oluşturulurken proje dosyasının (.sln uzantılı) projenin dosyaları ile aynı klasörde olmasını istersen bu seçenek işaretlenmemelidir.

## Visual Studio Panelleri

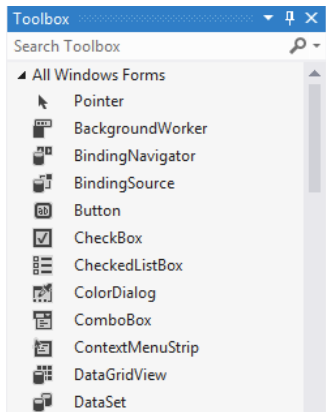
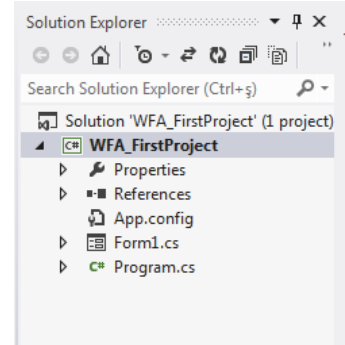
Yeni bir proje oluşturduktan sonra karşımıza visual studio panelleri gelecektir. Bu panelleri kullanarak projemizi şekillendirebilir ve kolaylıkla yönetebiliriz. Şu anda kullanacağımız paneller

- Solution Explorer
- Properties Window
- Toolbox

### Solution Explorer

Oluşturduğumuz projenin dosyalarını yönetmek için kullandığımız paneldir. Solution Explorer panelini kullanarak projemize yeni formlar, klasörler ve dosyalar ekleyebilir ve bu dosyaları yönetebiliriz.

Projemiz içerisinde ki referanslar ve tüm ayarlamalar bir klasörde tutulur. Bu klasörün içeriğini ise Solution Explorer penceresi ile yönetiriz. Yine Visual Studio'nun sunduğu bir çok özelliği Solution Explorer penceresi ile kullanabiliriz.



### Toolbox

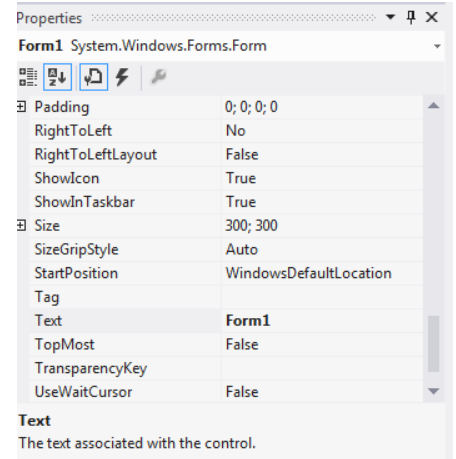
C Sharp'da kullanabileceğimiz kontrollerin bulunduğu araç kutusudur. Projemiz içerisinde kullanmak istediğimiz kontrolleri sürükleyip bırak yöntemiyle veya toolbox penceresinde kontrolün üzerine çift tıklama ile projemize eklenebilir. Projelerimizde kullanacağımız kontrollerin tamamı bu pencerede ki kontrollerden oluşmaktadır.

Toolbox içerisinde kendi sık kullandığımız kontrolün bulunduğu sekmeler ekleyebiliriz. Hatta 3. Parti yazılımların bize sunmuş olduğu extra kontrol ve componentleri eklememizi mümkün kılar.

## Properties Window

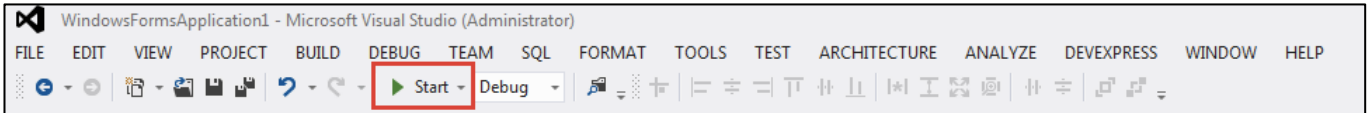
Kullandığımız kontrollerin özelliklerini değiştirebildiğimiz paneldir. Varsayılan değerleri barındırır. Kontrollerimize ait tüm özellikleri bu panelden değiştirebiliriz. Kolay kullanım açısından alınabilecek sabit değerleride listeler.

Properties penceresinden aynı zamanda kontrollerimizin Eventlerini de yönetebiliriz. Görsel ayar ve değişiklikleri yapabildiğimiz bu pencere Form penceremizdeki kontrollerin Form.Designer.cs dosyasındaki değişikliklerini yapar ve ordaki değerleri bize gösterir. Visual Studio içerisinde bulunan en önemli pencerelerden bir tanesidir. Pencerenin en altında seçilen özellik hakkında bilgilerde verilmektedir.



## Projeyi Derlemek ve Çalıştırmak

Şu ana kadar bir projeyi nasıl oluşturabildiğimizi gördük fakat henüz oluşturduğumuz projeyi çalıştırmadık. Projeyi çalıştırmak için F5 tuşunu kullanabiliriz. Proje çalıştırılma aşamasından önce Compile aşamasından geçecektir. Visual Studio içerisindeki Compiler yazdığımız C Sharp kodlarını Compile edecektir. F5 tuşunun yanı sıra visual studio panelinde bulunan 'Start' butonu ile de projelerimizi çalıştırabiliriz.



## PROGRAMLAMAYA BAŞLARKEN

Programlamaya başlamadan önce bilmemiz gereken terimler, klasörler ve kısayollar bulunmaktadır. Aşağıdaki tabloda döküman içerisinde sık karşılaşılabilecek terimler ve kısayolları bulabilirsiniz.

Tanım	Açıklama / Görevi
<b>App.config</b>	Bir proje oluşturduğunuzda Solution Explorer penceresi içerisinde, oluşturduğunuz proje altında Configuration File dosyası bulunmaktadır. Projemiz ile ilgili ayarların bulunduğu dosyadır. İleriki konularda Application Configuration dosyasının detaylarını inceleyeceğiz.
<b>References</b>	Solution Explorer içerisinden proje dosyalarınız arasında References bulunmaktadır. Projemiz içerisine yüklü kütüphane/dll dosyaları references içerisinde bulunmaktadır.
<b>Code Behind</b>	Program kodlarımızın bulunduğu sayfadır. Design sayfası ile bağlantılı olarak çalışmaktadır. Bir Windows Form projesi oluşturduğunuzda Form1'in üzerine sağ tıklayarak > View Code seçeneğini seçin. Açılan ekran kod yazmanız için gerekli Code Behind alanıdır.
<b>F4</b>	Properties penceresini açan kısayol tuşudur.
<b>F5</b>	Projeyi derlemek ve çalıştırmak için kullanılan kısayol tuşudur.
<b>F6</b>	Hazırlanan projeyi derlemek için kullanılan kısayol tuşudur.
<b>F7</b>	Hazırladığımız form'da Code Behind ekranına geçmek için kullanılan kısayol tuşudur.

**Ctrl + Shift + B** Projeyi derlemek için kullanılan kısayol tuş kombinasyonudur.

## Yazılımcı Dostu Bilgiler

C# programlama dilini daha etkin kullanabilmek için kullandığımız IDE'yi yani Visual Studioyu daha iyi tanımamız gerekmektedir. Aşağıda Hem C# hemde Visual Studio için yazılımcılara gerekli bilgiler bulunmaktadır.

### C#'da Yorum Satırı

C#'da kodlarımız yanlarına açıklama satırları ve notlar bırakmak isteyebiliriz. Bu notlar ileriye dönük geliştirmeye kolaylık sağlar. C# programlama dilinde yorum satırları koyu yeşil renktedir. Ayrıca yorum satırları program derlendiğinde ve çalıştırılmak istendiğinde çalıştırılmayan yazılardır. Program derlenirken ve çalıştırılırken hiç bir şekilde yorum satırları derleyici tarafından dikkate alınmaz.

Tek bir satır yorum yazmak istediğimizde // karakterleri yorum satırı yazacağımız anlamına gelir. Yorum satırları ile dikkat edilmesi gereken bir diğer nokta ise her programlama dilindeki yorum satırları farklı kullanılmaktadır. C# programlama dilindeki yorum satırları // karakterleri ile oluşturulmaktadır.

#### Tekli Satırdan Oluşan Yorum Yazmak

//Tekli yorum satırıdır. Tek satırda yorum yazmak istediğimizde kullanabiliriz.

C# programlama dilinde birden fazla satırda yorum yazmak istediğiniz durumlarda /\* karakterleri ile çoklu yorum satırını başlatabilirsiniz. \*/ karakterleri ise yorum satırlarının bittiğini göstermektedir. Bu iki ifade arasında yazdığınız tüm yazılar yorum satırı olarak görünmektedir.

#### Birden Fazla Satırda Yorum Yazmak

```
/*  
* Birden fazla satırda yorum yazmak : Satır 1  
* Birden fazla satırda yorum yazmak : Satır 2  
* Birden fazla satırda yorum yazmak : Satır 3  
*/
```

## Word Wrap

Visual C Sharp'da uzun kod satırları, yatay olarak sayfayı uzatır. Kod yazarken kod satırının tamamını görememek, geliştirme açısından hız kaybına sebep olabilir bu sebep ile satır sonuna geldiğimizde kodlarımızın alt satırdan devam etmesini isteyebiliriz. Bu özelliği açmak için aşağıdaki adımları takip edebilirsiniz.

- Visual Studio ana penceresinden Tools menüsünü açın.
- Option seçeneğini tıklayarak, Seçenekler penceresini açın.
- Sol ekrandan Text Editor sekmesinin solunda bulunan üçgen kıvrımına tıklayın.
- Açılan listeden All Languages seçeneğini seçin.
- Sağ tarafta bulunan seçeneklerden Word Wrap seçeneğini işaretleyin.

## Line Numbers

Bir programlama dili kullanarak geliştirme yaptığımızda kod satırlarının satır numaralarını görmek isteyebiliriz. Satır numaralarını görmek verilen hataları bulmamızı kolaylaştırmak gibi bir çok konuda yardımcı olur. Projemizde yazdığımız kod satırlarının satır numaralarını görmek için aşağıdaki yolu takip edebilirsiniz.

- Visual Studio ana penceresinden Tools menüsünü açın.
- Option seçeneğini tıklayarak, Seçenekler penceresini açın.
- Sol ekrandan Text Editor sekmesinin solunda bulunan üçgen kıvrımına tıklayın.
- Açılan listeden All Languages seçeneğini seçin.
- Sağ tarafta bulunan seçeneklerden Line Numbers seçeneğini işaretleyin.

## Scope

Programlarımızda kullandığımız yapıların kapsayacağı kodları belirler. Oluşturulan yapı çalıştığında, yapıya ait kodların nereden başladığı ve nerede bittiğini belirler. C Sharp'da scope alanları süslü parantezler '{' içerisine kodlar '}' ile belirlenir. Farklı dillerde scope tanımlaması ve karakteri değişkenlik gösterebilir. Scopeları anlamak için aşağıdaki örneği inceleyebilirsiniz.

Bir proje oluşturduğumuzda Form1 dosyası açılmaktadır. Form1 yüklenirken yapılacak işlemleri Form1\_Load (Form1 Yüklenirken) eventi altına yazabiliriz. Form1\_Load eventine ait kodları scopelar arasına yazmamız gerekmektedir.

### Scopelar C# programlama dilinde süslü parantezler ile belirtilir

```
private void Form1_Load(object sender, EventArgs e)
{
    //Scope Başlangıcı

}
//Scope Bitişi
```

## Syntax

Syntax söz dizimi anlamına gelmektedir. Programlama dillerinde kullanacağımız kodların yapısı ve yazım şekli önemlidir. Kod sıralaması, parantez sıraları ve büyük küçük harf duyarlılıklarına syntax denir. C# syntax kurallarını aşağıda bulabilirsiniz.

- C# büyük küçük harf duyarlılığına sahiptir.
- C# programlama dilinde tüm anahtar sözcükler küçük harflerden oluşmaktadır.
- C# programlama dili ile program yazarken türkçe karakter kullanılmamalıdır.
- Proje ve kütüphaneler tarafında tanımlı özel kelimeler oluşturduğumuz yapılara verilemez.
- Oluşturulan dosyalarda türkçe karakterler kullanılmamalıdır.
- Bir kod satırı noktalı virgül ile sonlandırılır.
- Metotların, parametre parantezleri vardır.
- Kod yazarken komut kelimeler arası istenildiği kadar boşluk bırakılabilir.
- Kod yazarken satırlar arası istenildiği kadar boş satır bırakılabilir.



## Indent Guides

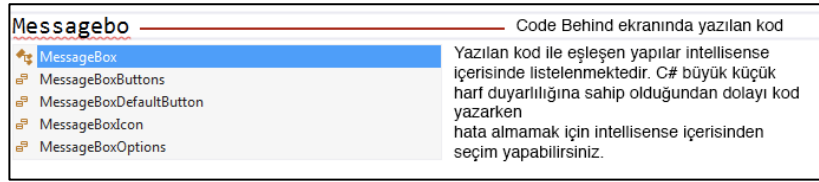
Visual Studio ortamına daha kolay geliştirme yapmak için eklentiler kurabilirsiniz. Bu eklentiler sizin daha kolay proje geliştirmenize ve kodlarınızı daha kolay okumanıza olanak sağlar. Indent Guides bize scope başlangıç ve bitişlerini gösteren ve açılıp-kapanan scope'ları noktalı çizgiler ile birbirine bağlayan bir eklentidir. Indent Guides kurmak için aşağıdaki adımları takip edebilirsiniz. Sağ tarafta bulunan resimde scope başlangıç ve bitişleri ..... ile işaretlenmiştir. Indent Guides kurmak için;

- Visual Studio sekmelerinden Tools sekmesini açın.
- Extensions and Updates penceresini açın.
- Açılan pencerede sol panelde bulunan 'Online' seçeneğini seçin. Visual Studio Gallery içerisinde Indent Guides anahtar kelimesini aratın.
- Bulunan eklentinin yanında bulunan Install butonuna tıklayarak kurulum yapabilirsiniz.
- Kurulmdan sonra Visual Studio 2015'i yeniden başlatın.

## Intellisense

Yazılım geliştirici için en önemli ekranlardan bir tanesidir. Visual Studio'da kod yazarken, intellisense penceresi yazdığımız kodları hazır olarak tamamlayan, kullanılabilecek kodları listeleyen ve yer alan kod yapıları hakkında bilgi veren bir penceredir. Kod yazmayı kolaylaştırır, hızlı ve etkili bir biçimde kod yazılmasını sağlar. Intellisense visual studio içerisinde varsayılan olarak bulunan bir penceredir.

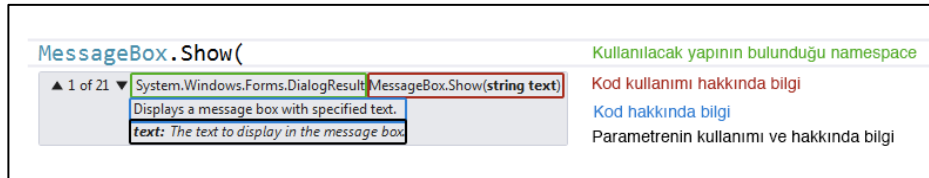
Intellisense'i kullanmak için code behind ekranında klavyeden bir harfe basıldığında bu harfi içeren ve bu harf ile başlayan komutları ekranda göstermektedir. Projemizde yüklü olan dll dosyalarındaki tüm yapıyı listelemektedir.



Intellisense penceresi kapandığında, CTRL + Space tuş kombinasyonu ile tekrar açabilirsiniz.

## Tooltip

Bir yazılım geliştirici için en önemli ekranlardan bir tanesidir. Program geliştirirken yazılan kodların nasıl kullanması gerektiğini ve kullanılacak yapının açıklamasını göstermektedir. Doğru kullanıldığı takdirde bir yazılımcı için en önemli penceredir. Aşağıdaki resimde tooltip kullanımı hakkında detaylı bilgi bulabilirsiniz.



## Smart Tag

Bazı kontrollerin çok kullanılan özelliklerini Smart Tag penceresinde bulunur.. Belirli kontrollerin en çok kullanılan özelliklerine daha hızlı erişim kolaylığı sağlar.

## YAZILIMA GİRİŞ

Artık yavaş yavaş kod tarafına geçme vaktimiz geldi. İlk kod örneğimizi yazmadan önce bir takım kurallardan bilmemiz gerekiyor. C Sharp'da dikkat edilmesi gereken kurallar;

(Sağ tarafta bir formun code ekranına geçtiğinizde karşılaşıacağınız yapıyı görmektesiniz.)

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
    }
}
```

- C# programlama dilinde, satır sonları ';' ile işaretlenir. Bu işaret kod satırının bittiği anlamına gelir.
- C# dilinde söz dizimine Syntax denir. Syntax dilin belirli kurallarının olması ve kod yazarken bu kurallara uyulması gerektirir. Eğer biz bu kurallara uymadan kodlarımızı yazmaya çalışırsak Syntax hataları alırız. İleri ki konularda Syntax'ı detaylı inceleyeceğiz.
- C# Programlama dili case sensity (büyük küçük harf duyarlılığı) bir dildir. Büyük küçük harf duyarlılığı ASCII kodlarından kaynaklanmaktadır. Klavyeden girilen her tuşun bir ASCII karşılığı bulunmaktadır. Büyük ve küçük harflerin ASCII kodları farklı olduğundan programlama yapılırken büyük küçük harf ile tanımlanan değerler farklı değerlerdir.
- C# programlama dilinde projeleri ve dosyaları isimlendirirken tanımlı olan özel kelimeleri ve isimleri dosya ve proje isimlerinizde kullanılmamalıdır.

## Events (Olaylar)

Eventler, bir kontrolü kullanmamız için gereken zaman kavramlarıdır. Kullanacağımız kontrolün zaman kavramlarını yöneterek kod yazabiliriz.

Bizler kontrolün zaman kavramlarını çok iyi bilmeliyiz. Örneğin bir TextBox'ın eventlerinden bahsedecek olursak;

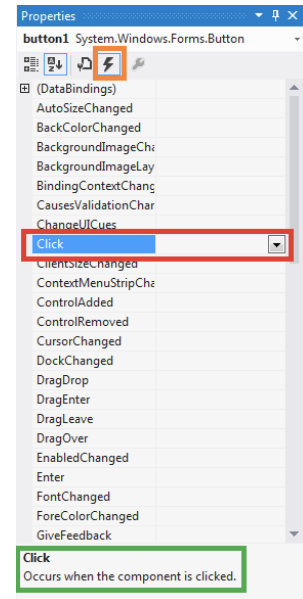
**TextChanged** : Textbox içerisinde girilen yazı değiştiğinde hangi kodların çalışacağı yazılır.

**MouseDoubleClick** : Mouse ile çift tıklandığında hangi kodların çalışacağı yazılır.

**MouseMove** : Mouse ile Textbox üzerinde hareket edildiğinde yapılacak kodlar yazılır.

Yukarıda bahsettiğimiz kontrollerdin eventlerinde zaman kavramına dikkat etmemiz gerekir. Altı çizili olan yazılarda bir zaman kavramı bulunmaktadır.

Event Driven Programming (Olay Güdümlü Programlama) yapılan dillerde zaman kavramı tarih ve saat olarak değil, bir olayın gerçekleşmesi olarak algılanır. (Örnek olarak; butona tıklandığı (button click) zaman..)



## Event Mantığı

Bir programcı zaman kavramlarını çok iyi biliyor ve bu zaman kavramlarını çok iyi yönetebiliyor olmalıdır.

Programlamanın temelinde iki soru vardır. Bu iki sorunun mantığı tam olarak kavrandığında kod yazmak çok kolaylaşacaktır.

Soru 1 : Ne zaman yapacağım?

Soru 2 : Ne yapacağım?

Yukarıdaki soruları bir örnek ile açıklayalım.

Soru 1 : Ne zaman yapacağım? Örnek Cevap : Butona tıklandığı zaman

Soru 2 : Ne yapacağım? Örnek Cevap : Kullanıcıya mesaj göstereceğim.

Yukarıdaki soru ve cevap örneğini incelemek için aşağıdaki adımları uygulayabiliriz.

**Örnek Kod :** Kod yazmaya başlamadan önce yeni bir proje oluşturmamız gerekiyor. Visual Studio programını çalıştırdıktan sonra, yeni bir proje oluşturuyoruz. Proje oluşturma hakkında detaylı bilgiye önceki konularımızda değinmiştik. Windows Form Application olarak oluşturduğumuz proje de Form1 isminde bir form ekranı açılacaktır. Form1 ekranının Code Behind sayfasına ulaşmak için F7 tuşuna basmamız yeterlidir.

İlk kodumuzu yazmak için formun Design alanına gelip form'a çift tıklıyoruz. Forma çift tıkladığımızda bize Formun Event'lerinden Load eventini (Form Yüklenirken) Code Behind sayfasına oluşturur. Dikkat ettiyseniz Form Load eventinin Formun yüklenirken çalışacak olan eventi olarak bahsettik. Formun yüklenmesi sırasında işleyecek kodlar load eventinde yazılır.

#### Form1\_Load Eventini oluşturuyoruz.

//Form'a çift tıkladığımızda aşağıdaki kodlar oluştur.

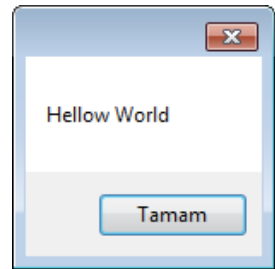
```
private void Form1_Load(object sender, EventArgs e)
{
}
```

#### Eventimiz içerisine Görüntülenecek yazımızı yapıştırıyoruz.

```
private void Form1_Load(object sender, EventArgs e)
{
    MessageBox.Show("Hello World");
}
```

Yukarıdaki kod satırı ekranda bir mesaj penceresi içerisinde "Hello World" yazısını yazdırır. Biz kullanıcıya mesajlar göstermek istediğimizde yukarıdaki kod satırını kullanabiliriz. Yukarıda ki kod satırını çalıştırdığımızda Resim 2.1'de bulunan görüntü gibi bir sonuca ulaşırız.

Kullanıcıya mesaj görüntülendikten sonra "Tamam" butonuna basarak bu pencereyi kapatabiliriz. *Burada ki buton isimlerinin Türkçe veya İngilizce olması işletim sisteminin diliyle ilgilidir.*



## Event Detayları

Her kontrole ait onlarca event bulunmaktadır. Bu eventleri çalıştırmak için ilgili kontrolü mouse ile seçtikten sonra Properties penceresinden Eventler sayfasına geçmeliyiz.

Events sekmesinde listelenen eventler, o kontrole ait olan event listesidir. Bizler bir kontrolün zaman kavramını eventler ile yönetebiliriz. (Örneğin textbox'a veri girildiği zaman)

Herhangi bir kontrolün üzerine çift tıkladığımızda, Visual Studio tarafından tanımlı olarak o kontrole ait en çok kullanılan event Code Behind sayfamızda oluşur.

Event'ler genellikle zaman için kullanılan bir kavram. Örneğin bir düğmenin tıklanması bir event oluşturur. Olay güdümlü programlama dillerinin temelini oluştururlar. Eventler, programlamada ki zaman kavramlarıdır. Butonun tıklanması, formun yüklenmesi gibi zamanlarda yapılacak işlemleri yönetirler. Programı zaman aralıklarına göre yönetebiliriz, kullanıcının işlem yaptığı zamanları programlayarak işlemlerimizi yapabiliriz. C Sharp'da en çok kullanılan eventler ve açıklamaları aşağıdadır.

Event	Açıklama
<b>Click</b>	Kullanıcının kontrole tıkladığı zaman aralığında tetiklenen eventtir.
<b>DoubleClick</b>	Kullanıcının kontrole çift tıklığı zaman aralığında tetiklenen eventtir.
<b>MouseClick</b>	Kullanıcının kontrole mouse ile tıkladığı zaman aralığında tetiklenen eventtir.
<b>Mouse Double Click</b>	Kullanıcının kontrole mouse ile çift tıkladığı zaman aralığında tetiklenen eventtir.
<b>Mouse Move</b>	Kullanıcının ilgili kontrol üzerinde fare imlecini hareket ettirdiğinde tetiklenen eventtir.
<b>Mouse Enter</b>	Kullanıcının ilgili kontrol üzerine mouse ile geldiğinde tetiklenen eventtir.
<b>Mouse Leave</b>	Kullanıcının kontrol üzerinden ayrıldığında tetiklenen eventtir.
<b>Mouse Down</b>	Mouse tuşuna basıldığında tetiklenir. Click olayından önce çalışan bir olaydır.
<b>Selected Index Changed</b>	ComboBox, Listbox gibi kontrollerde seçili index değiştiğinde tetiklenen eventtir.

## Atama Operatörü

Yazıldığı yerin sağında bulunan değer, eşittirin soluna aktarılır. Yukarıdaki örnekte Textbox kontrolünün Text özelliğinde ki (Property) değer string tipindeki değişkenimize aktarılmıştır. Atama Operatöründe dikkat edilmesi gereken çok önemli bir nokta vardır. SADECE EŞİTLENEBİLİR DEĞERLER BİRBİRİNE EŞİTLENİR.

Yani; atama operatörünün solunda bulunan yapı string bir değer istiyorsa, atama operatörünün sağında bulunan değer ise bir string değer taşıyorsa atama işlemi gerçekleştirilir. Eğer atama operatörünün sağında ve solunda bulunan değerler birbirine eşit değilse atama gerçekleştirilemez ve bize Syntax hatası verir.

İki değer farklı olması durumunda dönüştürme işlemlerini kullanarak iki değer arasında atama gerçekleştirebilir. Dönüştürme işlemlerini sonraki konularda inceleyeceğiz.

Event mantığını kullanarak küçük bir alıştırmayı yapalım ve alıştırmamız sırasında atama operatörlerini inceleyelim. Çok basit bir örnek ile başlayalım. Hazırladığımız projede yapmak istediğimiz, Checkbox kontrolüne tıklanıkça butonun aktifliğini değiştirmektir. Biz checkbox kontrolünü işaretlediğimizde buton aktif olacak, işareti kaldırdığımızda ise pasif olacaktır.

Bu işlemi yapabilmek için öncelikle checkbox kontrolünün seçiminin değiştiği zamanı kontrol eden eventi bulmalıyız. Checkbox kontrolünde bu işi üstlenen event "CheckedChanged" eventidir. Checkbox işaretlendiğinde ve işaret kaldırıldığında bu event çalışır.

Checkbox kullanıcıdan seçimler yapmasını istediğimizde kullandığımız bir kontroldür. Aşağıdaki kodları Checkbox kontrolümüzün CheckedChanged eventine yazmamız gerekmektedir.

### Checkbox'ın CheckedChanged Eventini Oluşturuyoruz

```
private void chkAktifMi_CheckedChanged(object sender, EventArgs e)
{
    btnTikla.Enabled = chkAktifMi.Checked;
}
```

Eşittir '=' C Sharp'da atama operatörü olarak kullanılmaktadır. Sağdaki değeri sol tarafa atar. **Buradaki önemli nokta birbirine eşitlenebilir değerler birbirine eşitlenebilir.**

## VARIABLES (DEĞİŞKENLER)

Bizler projelerimizde her zaman sabit verilerle çalışmayız. Kullanıcıdan veri almamız gereken durumlar olabilir veya biz projemiz çalışırken bazı verileri değiştirmek isteyebiliriz. İşte böyle bir durumda kullanabileceğimiz yapılar olan "Değişken"leri kullanırız.

Değişken, verilerimizi ram üzerinde saklayan alanlardır. Biz bilgisayarlarımızda yaptığımız her işlemi RAM isimli parça üzerinden yaparız.

### Random Access Memory (RAM)

Bilgisayarın donanım bileşenlerindendir. Mikroişlemcili sistemlerde kullanılan bür tür veri deposudur. RAM geçici bir hafıza birimidir. Bilgisayar çalıştığı süre boyunca üzerinde veri tutabilir. RAM'de ki güç kesildiğinde (Bilgisayar kapatıldığında) RAM, üzerinde bulunan tüm bilgileri kaybeder.

RAM'in amacı bizler bilgisayarlarımızda işlem yaparken, Hard Disk gibi yapıların yavaş çalışmasından dolayı, işlemlerde bekleme yaşamamamız için vardır. RAM yapı olarak yüksek hıza sahip olan donanımlardır.

C Sharp'ta tipler Reference Type ve Value Type olarak ikiye ayrılır. Bunlara veri tipleri denir. Veri tipleri, bellekte ayrılacak bölgenin büyüklüğünü belirlemek için kullanılır.

Bir programcının bir dili öğrenmeye başladığında ilk öğrenmesi gereken o dile ait veri tipleridir. Çünkü veri tipleri program içerisinde kullanılacak değişkenlerin ve sabitlerin sınırlarını belirler. C Sharp programlama dilinde veri tipleri türlere ayrılır.

## Veri Tipleri

C# programlama dilinde sayısal, metinsel, mantıksal gibi bir çok farklı kategoride veri tipi bulunmaktadır. Aşağıda C# programlama dilinde bulunan veri tiplerinin incelemesini bulabilirsiniz.

### Sayısal Türler

RAM bellek üzerinde sayısal işlemler yapılmak üzere sayısal veriler tutmak istediğimizde kullanacağımız veri tipleridir. Yandaki resimde sayısal değişken türlerinin tablosunu bulabilirsiniz. Tabloda;

**Tür** : Oluşturacağınız değişkenin tipi

**Boyut** : Ram üzerinde kapladığı alan

**Kapasite** : Tutacağımız değer aralığı

Örnek : Kullanmak için gereken kod satırı şeklinde gösterilmektedir.

Tür	Boyut	Açıklama	Örnek
byte	1 byte	0 ile 255 arası tam sayı	<code>byte a = 5;</code>
sbyte	1 byte	-128 ile 127 arası tam sayı	<code>sbyte a = 5;</code>
short	2 byte	-32768 ile 32767 arası tam sayı	<code>short a = 5;</code>
ushort	2 byte	0 ile 65535 arası tam sayı	<code>ushort a = 5;</code>
int	4 byte	-2147483648 ile 2147483647 arası tam sayı	<code>int a = 5;</code>
uint	4 byte	0 ile 4294967295 arası tam sayı	<code>uint a = 5;</code>
long	8 byte	-9223372036854775808 ile 9223372036854775807 arası tam sayı	<code>long a = 5;</code>
ulong	8 byte	0 ile 18446744073709551615 arası tam sayı	<code>ulong a = 5;</code>
float	4 byte	-3.402823E38 ile +3.402823E38 arasında	<code>float a = 5f;</code>

doubl e	8 byte	-1.79769313486232E308 ile +1.79769313486232E308 arasında	<code>double a = 5d;</code>
deci mal	16 byte	-7.9E1028 ile +7.9E1028 arasında	<code>decimal a = 5m;</code>

## Metinsel Türler

Bellek üzerinde metinsel ifadeler saklamak için kullanılan değişken tipleridir. İki çeşit metinsel veri tipi vardır. Sınırsız karakter tutma yeteneğine sahip string ve tek bir karakter tutma yeteneğine sahip char veri tipinin özelliklerini yanda ki tablodan inceleyebilirsiniz.

Tür	Boyut	Açıklama	Örnek
char	2 byte	U + u + ffff 0000 unicode karakter.	<code>char a = 'X'; // Character literal</code> <code>char b = '\x0058'; // Hexadecimal</code> <code>char c = (char)88; // Integral Type</code> <code>char d = '\u0058'; // Unicode</code>
string	Sınırsız	Sınırsız boyutta veri saklar.	<code>string metin = "Metinsel İfade";</code>

## Mantıksal Türler

Mantıksal işlem sonuçlarını saklamak için 'boolean' ismiyle kullanılan kullanılan bir veri tipi bulunmaktadır. Koşullu yapılarda kullanılır ve koşul sonucunu saklayabilir.

Tür	Açıklama	Örnek
Boolean	True veya False Değer	<code>//Yıl, Gün, Ay Formatında tarih tipinde değer oluşturabileceğiniz gibi saat dakika saniye formatlarında oluşturulan zaman formatına eklemek mümkündür.</code> <code>DateTime zaman = new DateTime(1991, 6, 9);</code>

Bool veri tipi içerisinde sadece tek değer saklayabilir. İçerisinde true veya false veri tutabilir. Bu veri tipini mantıksal şartlar sağlandığında kullanabiliriz.

## Object Veri Tipi

Bu değişken türüne her türden veri saklanabilir. Object veri tipine veri atama işlemine boxing denir. Object veri tipi içerisinde bulunan veriyi dışarı çıkartma işlemine ise unboxing ismi verilmektedir. Veri ram üzerinde object tipte saklanmaktadır.

### Boxing ve Unboxing İşlemi

//Object veri tipi içerisinde istediğiniz tipte değer saklayabilirsiniz. Bu işleme boxing denir.

`object metin = "Metinsel İfade";`

`object sayi = 5;`

`object ondalikli = 10.24m;`

`object mantiksal = true;`

`object karakter = 'C';`

//Object içerisinde, içinde saklanan veriyi elde etmek için unboxing işlemi yapılmaktadır.

`int eldeEdilenSayi = (int)sayi;`

## Var Kullanımı

C# 3.0 da artık tür belirtmeksizin değişken tanımlamamıza olanak sağlayan yenilikler mevcut. Ancak bu değişkenlerin özelliği object değişkenler gibi referans tipli değişkenler değildir. Bunun yerine değeri atanırken tipinin belirlendiği değişkenlerdir.

### var anahtar kelimeni kullanmak

//var anahtar kelimesini kullanarak değer sakladığınızda, var ifadesi içerisine attığınız verinin tipine göre ram üzerinde veri tipini belirlemektedir. var ifadesinde object olarak oluşturulmadığından örneğin içerisine bir string ifade attığınızda hiç bir convert işlemi yapmadan string metotlarını kullanabilirsiniz. Diğer veri tipleri içinde aynı durum geçerlidir.

```
var deger = "Ahmet";  
deger.ToLower();
```

## Datetime (Tarih Zaman)

İçinde zaman barındıran value type değişken tipidir. Gün, Ay, Yıl ve diğer zaman bilgilerini bir arada tutarak zaman formatında işlemler yapmamızı sağlar. DateTime.Now komutu ile sistemin o an ki tarih ve saat bilgisini okuyabilmekteyiz.

Tür	Boyut	Açıklama	Örnek
DateTime	8 byte	Tarih ve Zaman Tutar	<code>DateTime zaman = new DateTime(1991, 6, 9);</code>

## Local ve Global Değişkenler

C Sharp'da oluşturduğumuz yapıların scope alanlarına göre erişilebilirlik durumları vardır. Bunlara yaşam alanı denir. Bir event içerisinde tanımlanan bir değişkene başka bir event içerisinden erişemeyiz. Bir değişkene birden fazla event içerisinde erişmek istediğimizde değişkeni scope alanları dışarısında yani global alana tanımlamamız gerekmektedir.

Global alan Form içerisinde, eventleri, metotları ve diğer yapıları tanımladığımız, proje derlendiğinde RAM belleğe ilk çıkartılacak yapıları belirleyen zaman kavramından uzak yapıdır. İçerisinde eventler, metotlar ve proje derlendiğinde RAM üzerine çıkartılmasını istediğimiz yapıları tanımlayabiliriz.

## Local Değişkenler

Belirli bir event veya metoda bağlı değişkenlerdir. Yaşam alanları (erişilebilecek yer) sadece bağlı oldukları scope alanlarıyla sınırlıdır. Örneğin Formun Load eventinde tanımladığımız bir değişkene başka bir eventten erişemeyiz.

### Local Değişkenler

```
//Formun Load Eventini Oluşturuyoruz.  
private void Form1_Load(object sender, EventArgs e)  
{  
    //Local Değişken Tanımlama  
    //string Tipinde Değişken oluşturma ve değer atama  
    string programlamaDili = "C Sharp";  
    //char Tipinde Değişken oluşturma ve değer atama  
    char harf = 'A';  
    //int Tipinde Değişken oluşturma ve değer atama
```



```
int sayi = 25;

//double Tipinde Değişken oluşturma ve değer atama
double ondalikliSayi = 10.5;

//decimal Tipinde Değişken oluşturma ve değer atama
decimal fiyat = 55.90m;

//bool Tipinde Değişken oluşturma ve değer atama
bool bilgisayarAcikmi = true;

}
```

String tipde değişkenler değer olarak içerisinde alfanumerik karakter toplulukları alabilir. C Sharp programlama dilinde string ifadeler çift tırnak içerisinde yazılır.

Char tipindeki değişkenler tek bir karakteri barındırabilir. Char tipde bir değişken tanımlandığında tek tırnak arasında istediğimiz alfanumerik bir karakteri değer olarak atayabiliriz.

Sayısal tipdeki (int, float, double, long, uint, ulong vs.) değişkenler üzerinde sadece sayısal bilgileri tutabilir. Dikkat edilmesi gereken bir nokta ise sayılar aynı zamanda alfanumerik olduğu için string tipi içerisinde tutulabilir. Fakat sayı ile bir matematiksel işlem yapılıyorsa değişkenin tipini sayısal bir tip olarak tanımlamalıyız.

Bool tipindeki değişkenler yalnızca iki değer alır. True ve False değerleri.

Double ve decimal tipindeki değişkenler içerisinde ondalıklı sayı saklayabiliriz. Decimal tipinde ondalıklı sayı verdiğimizde sayının sonuna 'm' karakteri eklenir.

DateTime tipindeki değişkenler içerisinde tarih formatında bilgi saklamaktadır.

## Global Değişkenler

Bir değişkene, birden fazla event'den veya metotdan ulaşmak istediğimizde tanımlama yerini değiştirmemiz gerekebilir. Global değişkenler proje derlenirken RAM bellek üzerine çıkartılır ve program sonlanana kadar RAM üzerinde yaşamaya devam ederler. Global değişkenlere Form içerisinde birden fazla yerden erişilebilir.

### Global Değişken Tanımlama

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    //Global Değişken Tanımlama
    string globalDegisken = "Bu Global Bir Değişkendir.";

    //Forma çift tıklayarak Load Eventini oluşturabiliriz.
    private void Form1_Load(object sender, EventArgs e)
    {
        globalDegisken = "Formun Load Eventinden Global'de Tanımlanan Değişkene Erişebiliriz.";
    }

    //Form üzerine bir adet button ekleyip, üzerine çift tıklayarak Buttonun Click eventini oluşturabiliriz.
    private void button1_Click(object sender, EventArgs e)
```

```
{  
    globalDegisken = "Butunun Click Eventinden Global'de Tanımlanan Değişkene Erişebiliriz.";  
}  
}
```

## Constant (Sabitler)

Uygulamalarımızda uygulama başlamadan önce (geliştirme aşamasında) bir değer atayıp daha sonradan kodlar içerisinde bu değerın RAM bellek üzerinde değiştirilmeden saklanmasını sağlayabiliriz. Bunun için oluşturduğumuz değişkeni const anahtar kelimesi ile yazarız. Bu anahtar kelime değişkene değer atandıktan sonra bir daha bu değerin değiştirilmemesini sağlayacaktır. Constant tanımlamak için aşağıdaki gibi tanımlama yapabilirsiniz.

### Sabit Tanımlama Örneği

```
const string degiskenAdi = "Bu Bir Değişkendir.";
```

## CONVERT İŞLEMLERİ

Veri tipler arasında dönüşüm yapmak için kullanılır. Örnek vermemiz gerekirse C# programlama dilinde string ifadeler “çift tırnak” arasında yazılır. Aynı zamanda “23” de bir string ifadedir. Ama 23 aynı zamanda bir sayıdır. 23 ifadesini çift tırnak içerisine aldığımızda string ifade elde etmiş oluruz. Ancak biz 23 sayısını bir matematiksel işleme tabi tutacağımız zaman, 23 değerini sayı olarak elde etmeliyiz.

Kullanıcı Textbox kontrolünü kullanarak bir sayı girişi yaptığında, bu değer C# tarafına string bir ifade olarak yansır. Bizim bu ifadeyi integer bir ifadeye dönüştürmemiz gerekir.

### Dönüştürme İşlemi

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

Yukarıdaki kod textboxdan gelen veriyi, tipi int olan değişkenin içine aktarır. Eğer Textbox'ın Text Özelliğinde (kullanıcının girdiği değer) bir sayı ise yukarıda ki kod bloğu hatasız olarak çalıştırılacak ve kullanıcının string tipte girdiği sayıyı integer(sayısal) olarak elde etmiş olacağız.

## EXCEPTION HANDLING

Hata yönetimidir. Yazılımcılar farklı türlerde, bir çok hata ile karşılaşabilirler. Önemli olan nokta karşılaştığımız hataları çözebilmek ve bu hataları kısa sürede doğru bir şekilde kontrol altına alabilmektir. Bir hata ile karşılaştığımızda bu hataların öncelikle ne tür bir hata olduğunu bilmemiz gerekmektedir. Hatalar kendi içerisinde gruplara ayrılır.

### Syntax Errors (Derleme Zamanı Hataları)

Syntax hataları, bir yazılımcının en çok karşılaşacağı hata çeşitidir. Hatayı bulmak ve yönetmek kolaydır. Error List penceresinde hata mesajı görünür. Syntax Hatalarını düzeltmediğimiz zaman program çalışmayacaktır.

Gelişmiş bir IDE olan Visual Studio anlık olarak kodlarımızı kontrol edecek ve bize yardım edecektir.

### Runtime Hataları (Çalışma Zamanı Hataları)

Programda bir syntax hatası olmamasına karşı, kullanıcının yanlış bilgi girmesi veya programcının kontrol altına almadığı bir durum sonucu karşılaşılan hatalardır. Bu tür hatalardan korunmak için kullanılan yapılar bulunmaktadır.

## Logical Hatalar – Mantıksal Hatalar

Mantıksal hatalar, tespit edilmesi en zor hata çeşitidir. Syntax olarak hata vermemekle birlikte, runtime hataları gibi programda bozukluğa sebep olmaz. Yazdığımız kodlar istediğimiz sonucu döndürmüyorsa Mantıksal bir hata ile karşı karşıya olabiliriz. Mantıksal hataları tespit etmek için Runtime sırasında kodlarımızı adım adım kontrol etmemiz gerekebilir.

## Hata Yönetimi

Hata çeşitlerini incelediğimize göre şimdi sırasıyla bu hataları nasıl yönetebilir ve kontrol altına alabiliriz bunu inceleyelim. İlk olarak hatalar içerisinde en kolay tespit edilen grup olan Syntax Hatalarını nasıl giderebileceğimizi inceleyelim.

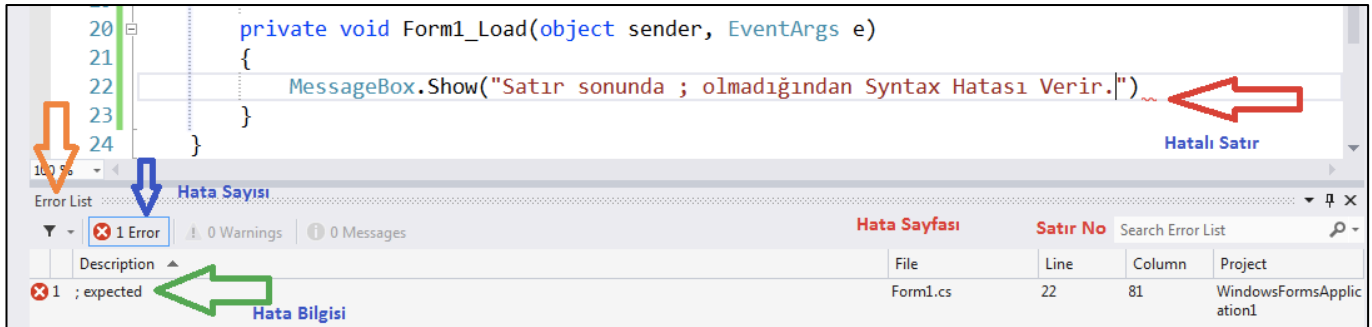
Projeyi derlediğimizde proje içerisinde bir syntax hatası bulunuyorsa (noktalı virgül unutmak, scope kapatmamak, yanlış değerleri eşitlemek vb.) Error List penceresinde Syntax hatalarımız görünür. Yapmamız gereken, hata satırını okumak ve hata satırı üzerine çift tıklayarak hata veren kod satırını görüntülemektir.

## Error List

Syntax hatalarımızı görüntüleyebileceğimiz penceredir. Projeyi derlediğimiz zaman projede ki syntax hataları Error List penceresinde görüntülenecektir.

İlgili hatanın üzerine çift tıklayarak hata veren satırı (Kırmızı ok ile işaretlenmiş) görüntüleyebiliriz.

Verilen hata mesajına göre (Yeşil ok ile işaretlenmiş) hatamızı düzeltebiliriz. Error List penceresi bize projemizde kaç adet hata olduğu (Lacivert ok ile işaretlenmiş) bilgisinde verir.



Error List penceresinde hatalarımız dışında Warnings sekmesi altında uyarılarda görünür. Warnings sekmesinde çıkan mesajlar hata değildir. Projemiz içerisinde bulunan kodlar hakkında bize uyarı verirler. Örneğin bir değişken oluşturduysak ve bu değişkeni kullanmadıysak, RAM'de gereksiz yer kapladığı bilgisini verir. Projenizin çalışmasına engel değildir.

## Try Catch Finally

Runtime hatalarını yönetmek için kullanılır. Yazılımcının ön göremediği durumlarda hatayı engellemek veya kullanıcı kaynaklı hataları kontrol altına almak için kullanılan yapıdır.

Try Catch Finally bloklarının çalışma mantığı, Try blokları içerisinde bulunan kodlarda herhangi bir sebepten hata olursa, Try bloğundaki kod akışı durur ve kod akışı Catch bloğundan devam eder. Catch bloğundaki kodlar işlendikten sonra Finally bloğu çalıştırılır. Genellikle Finally bloğunda temizleme, form düzenleme gibi işlemler yapılır.

Try Catch Finally yapısında Try ve Finally bloğu mutlaka çalışır. Catch bloğu yalnızca Try bloğunda oluşacak hata sonucunda çalışacaktır.

Try Catch Finally yapısında Finally bloğunu kullanmak isteğe bağlıdır. Programcı dilerse Try Catch blokları ilede çalışabilir.

**Try Catch Finally Blokları**

```
try
{
    string deger = "5";

    int sayi = Convert.ToInt32(deger); //Eğer gelecek değer bir sayı ise dönüştürme işlemi başarıyla gerçekleşecektir. Eğer deger değişkeninden
    gelen bir harf veya farklı bir karakter ise kod bloğu hata verecektir.
}
catch
{
    //deger değişkeninden bir harf veya özel karakter geldiğinde catch bloğu çalışmaya başlar.
    MessageBox.Show("Lütfen sayı girin.");
}
finally
{
    //Form Temizleme
    //Textbox Odaklanma
    //gibi işlemler yapılır.
}
```

**Try Catch Kullanımı – Hatayı Özelleştirme**

```
try
{
    double sayi1 = Convert.ToInt32(txtSayi1.Text);
    double sayi2 = Convert.ToInt32(txtSayi2.Text);
    double sonuc = sayi1 / sayi2;
    lblSonuc.Text = sonuc.ToString();
}
catch (FormatException ex) // Format Hatası olduğunda hata verir.
{
    MessageBox.Show("Lütfen doğru formatta veri giriniz.");
}
catch (OverflowException ex) //Değişken sınırları dışına çıktığında hata verir
{
    MessageBox.Show("Değişken sınırları dışına çıktınız.");
}
catch (DivideByZeroException ex) //Sayıları sıfıra bölmeye çalışırsan yakalar
{
    MessageBox.Show("Sıfıra bölünme hatası .. Sayı Sıfır 'a bölünemez");
}
catch (Exception ex) //Geri kalan tüm hataları denetler
{
}
```

```

    MessageBox.Show(ex.Message);
}

```

“Exception ex” kod satırındaki **ex** catch bloğunun özelliğidir.. Bu özellik içerisinde karşılaşılan hata ile ilgili tüm bilgileri tutar. `MessageBox.Show(ex.Message);` gibi bir komutla çalıştırdığımızda yakalanan hatanın mesajını kullanıcıya gösterir. Tüm catch bloklarında hataları **ex** ismi ile yakalıyoruz. Aynı isme sahip başka **ex**'ler olmasına rağmen diğer catch blokları ile çakışmamasının sebebi, her bir **ex** özelliği kendi scope'u ile ilgilidir. Kullanılan her **ex** özelliği kendi scopelarında çalıştığı için birbirleriyle

## Break Point

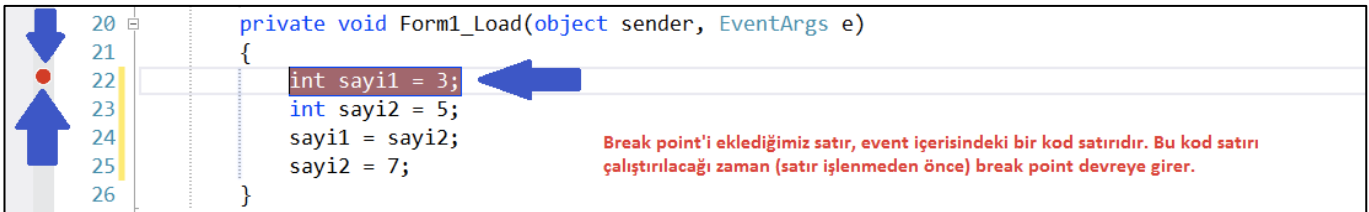
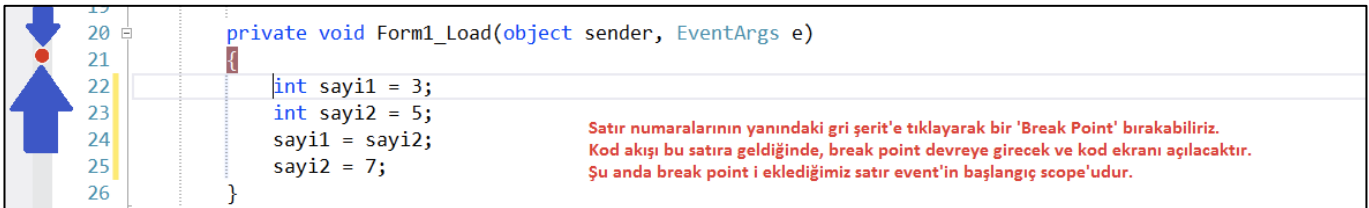
Break Point (Duraklama Noktası) debug işlemi sırasında sırasında kod akışını takip edebileceğimiz ve Runtime'da dönen değerleri program çalışırken izleyebileceğimiz özelliktir. Proje çalışırken verileri anlık olarak izlememize hangi kod satırında hangi işlem yapıldığını program çalışırken incelememize olanak tanır.

İzlemeye almak istediğimiz kod'u break point ile işaretlememiz gerekmektedir. Kod akışı sırasında, sıra break point ile izlemeye almak istediğimiz satıra gelince, visual studio'da ki kod ekranı açılacaktır. Bizler break point kısayolları ile değerleri program çalışırken inceleyebiliriz.

**F10** : Kod akışını takip etmeksizin izlemeye alır. Her tıkladığımızda satır satır kod akışı ilerleyecektir.


**F11** : Kod akışını takip ederek izlemeye alır. Uygulamadı ki kod akışını takip ederken, kod akışı farklı bir event veya metoda yöneliyorsa kod akışını izlemeye devam eder.

**Shift + F11** : Kod bloğunda ki akışı bitirir.



## Open Exception Helper

Exception Helper penceresi alınan hatalar hakkında detaylı bilgiler sunan ve hata çözümüne yönelik önerilerde bulunan bir ekrandır. Exception Helper penceresi etkin kullanıldığı takdirde hata tanıma çözme işlemleri kolaylaşacaktır.

 **FormatException was unhandled** **Hata Başlığı**

Input string was not in a correct format. **Hata Açıklaması**

**Troubleshooting tips:**  
When converting a string to DateTime, parse the string to take the date before putting each variable into the DateTime object.  
Make sure your method arguments are in the right format.  
Get general help for this exception.  
Search for more Help Online... **Hatayı Online olarak arar**

**Exception settings:**  
☐ Break when this exception type is thrown

**Actions:**  
[View Detail...](#) **Hatanın Detaylı Bilgisi**  
[Copy exception detail to the clipboard](#) **Hatanın detayını kopyalar**  
[Open exception settings](#) **Hata listesini açar**

## OPERATÖRLER

Matematiksel veya mantıksal işlem yapmamız gereken durumda operatörleri kullanabiliriz. Operatörler Matematiksel Operatörler, Mantıksal Operatörler, Arttırma Operatörleri, Birleşik Operatörler ve İlişkisel Operatörler olmak üzere gruplara ayrılır.

### Aritmetik Operatörler

Çeşitli matematiksel işlemleri yapmak için kullanılan operatörlerdir. Matematiksel operatörler arasında Toplama, çıkartma, çarpma, bölme ve mod alma gibi operatörler bulunmaktadır. Matematiksel operatörleri ve görevlerini aşağıdaki tabloda bulabilirsiniz.

Operatör	İşlem	Görevi
+	Toplama İşlemi	Verilen sayıları matematiksel toplama işlemine tabi tutar.
-	Çıkartma İşlemi	Verilen sayıları matematiksel çıkartma işlemine tabi tutar.
*	Çarpma İşlemi	Verilen sayıları matematiksel çarpma işlemine tabi tutar.
/	Bölme İşlemi	Verilen sayıları matematiksel bölme işlemine tabi tutar.
%	Mod Alma İşlemi	Sayının x'e bölümünden kalanı verir.

### Arttırma Operatörleri

Tam sayı tipinde tutulan değerlerin kısa komutlarla arttırılmasını ve azaltılmasını sağlayan operatörlerden oluşmaktadır. Kullanım detaylarını aşağıdaki tabloda görebilirsiniz.

Operatör	İşlem	Görevi
Sayı++	Toplama İşlemi	Yanına yazıldığı sayının değerini 1 arttırır. <u>Başka işlem varsa önce sayıyı arttırır</u> ve daha sonra işleme devam eder.
Sayı--	Çıkartma İşlemi	Yanına yazıldığı sayının değerini 1 azaltır. <u>Başka işlem varsa önce sayıyı azaltır</u> ve sonra işleme devam eder.
++Sayı	Çarpma İşlemi	Yanına yazıldığı sayının değerini 1 arttırır. <u>Başka işlem varsa önce o işlemi yapar ve sayıyı sonra arttırır.</u>

--Sayı	Bölme İşlemi	Yanına yazıldığı sayının değerini 1 azaltır. <u>Başka işlem varsa önce o işlemi yapar ve sonra sayıyı azaltır.</u>
--------	--------------	--

## Birleşik Operatörler

Birleşik operatörler bir matematiksel işlem operatörü ile atama operatörünün birleşmesinden oluşturulmuştur. Birleşik operatörlerin kullanım amacı kod satırlarını kısaltmaktır. Detayları aşağıdaki tablodan inceleyebilirsiniz.

Operatör	İşlem	Görevi
+=	Topla ve Aktar	Yanına yazıldığı sayının değerini 1 artırır. <u>Başka işlem varsa önce sayıyı artırır</u> ve daha sonra işleme devam eder.
-=	Çıkar ve Aktar	Yanına yazıldığı sayının değerini 1 azaltır. <u>Başka işlem varsa önce sayıyı azaltır</u> ve sonra işleme devam eder.
*=	Çarp ve Aktar	Yanına yazıldığı sayının değerini 1 artırır. <u>Başka işlem varsa önce o işlemi yapar ve sayıyı sonra artırır.</u>
/=	Böl ve Aktar	Yanına yazıldığı sayının değerini 1 azaltır. <u>Başka işlem varsa önce o işlemi yapar ve sonra sayıyı azaltır.</u>

## Mantıksal Operatörler

Programlarımız içerisinde mantıksal kararlar vermemiz gerekebilir. Belirli kriterleri mantıksal şartlar olarak inceleyip mantıksal sonuçlar çıkartmamız gereken durumlarda mantıksal operatörleri kullanabiliriz. Mantıksal operatörleri ve kullanım alanlarını aşağıdaki tablodan inceleyebilirsiniz.

Operatör	İşlem	Görevi
&	Ve	Bu Operatör her iki operandın değerlerini alır ve her iki operand'da doğru olmadığı sürece yanlış sonucunu üretir. Doğru sonucunu vermesi için her iki operandın doğru olması gerekir.
&&	Mutlak Ve	Normal & operatörlerinden tek farkı normal operatörlerin her iki operandı da önermeye dahil etmesiyken, && operatörünün sadece gerektiğinde diğer operandlara bakmasıdır.
	Veya	Herhangi birinin doğru olması, doğru sonucunu üretmek için yeterlidir. Sadece ve sadece iki operandın yanlış olması halinde sonuç yanlıştır.
	Mutlak Veya	Normal   operatörlerinden tek farkı normal operatörlerin her iki operandı da önermeye dahil etmesiyken, koşullu operatörlerin sadece gerektiğinde diğer operandlara bakmasıdır.
^	Özel Veya (Exclusive Or)	Bu operatör belirtilen iki koşulun doğru olması durumunda ve belirtilen iki koşulun da yanlış olması durumunda geriye false döner. Koşullardan sadece birisinin sağlanmasını istemektedir yani. Doğruluk tablosu aşağıdaki gibidir.

Özel Veya operatörü, sıkça kullanacağımız Mutlak Ve / Mutlak Veya operatörleri dışında pek sık kullanımına rastlayamayacağımız bir operatörümüzdür. Bu sebeple Özel (^) Veya operatörünü yakından inceleyelim.

Koşul1	Koşul2	^ (EXCLUSIVE OR)
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE



## İlişkisel Operatörleri

İlişkisel operatörler boolean sonuç döndürür. İki operand'ın birbirleriyle olan ilişkilerini kontrol ederek mantıksal sonuçlar üretmektedir.

Operatör	İşlem	Görevi
==	Eşit Mi	Verilen iki değer eşit olma durumunu kontrol eder.
!=	Eşit Değilmi	Verilen iki değer eşit olmama durumunu kontrol eder.
>	Büyük Mü	Birinci değer, ikinci değerden büyük olma durumunu kontrol eder.
<	Küçük Mü	Birinci değer, ikinci değerden küçük olma durumunu kontrol eder.
>=	Büyük Eşit Mi	Birinci değer, ikinci değerden büyük veya eşit olma durumunu kontrol eder.
<=	Küçük Eşit Mi	Birinci değer, ikinci değerden küçük veya eşit olma durumunu kontrol eder.

## Random Sınıfı

Rasgele sayı üretmemiz gereken durumlarda kullanabileceğimiz sınıftır. Rasgele sayı üretme işlemi, bilgisayardan veriler toplayarak bu verilerin belirli bir algorithmadan geçirilerek sayı doğrusu üzerinde belirttiğimiz değerler arasında bize bir sayı üretmesi sonucuyla oluşur.

Bilgisayar biliminde rasgele diye bir işlem yoktur. Tüm bilgi ve veriler belirli bir algoritma sonucu üretilir. Rasgele dediğimiz işlem bilgisayardan toplanan bilgilerin (Bilgisayarımızın, ram doluluğu, işlemci ısısı, ram çalışma hızı, hard disk doluluk oranı, ekranın yinleme hızı vs.) algorithmaya alınması ve sonuç üretmesi durumudur.

### Random Kullanımı

```
private void button1_Click(object sender, EventArgs e)
{
    Random rnd = new Random();
    int rasgelesayi = rnd.Next(); //Rasgele bir sayı üretir.
    int sayi1 = rnd.Next(50); //0 ile verdiğimiz değer arası bir sayı üretir.
    int sayi2 = rnd.Next(20, 100); //Belirttiğimiz iki değer arası sayı üretir.
}
```

## DateTime Sınıfı

Tarih ve saat formatında bilgi tutabileceğimiz bir yapıdır. Çeşitli formatlarda tarih bilgisi tutabiliriz. Tarih bilgisinin içerisinden istediğimiz değeri çekebilir, istediğimiz formatta bilgileri gösterebiliriz. DateTime sınıfında veri tutmak için;

### Datetime Kullanım Örneği

```
private void Form1_Load(object sender, EventArgs e)
{
    DateTime TarihSaat = DateTime.Now;
}
```

Yukarıdaki kod bloğu formun açıldığı tarih ve saati, RAM üzerinde tarih ve saat tipinde tutar. DateTime TarihSaat; ram üzerinde tipi DateTime olan bir TarihSaat nesnesi oluşturur. DateTime.Now komutu ise sistemin şu anki saatini döndürmektedir.

## Timer Kontrolü

Milisaniye cinsinden belirtilen zaman aralıklarında kodlarımızı çalıştırmak için kullandığımız kontroldür. Örneğin bir işlemin 10 saniyede bir kez yapılmasını istediğimizde kullanabiliriz. Bu işlemi yapmak için kullanıcının herhangi bir butona tıklaması gerekmez. Timer kontrolü otomatik olarak 10 saniyede bir çalışacaktır. İçeriğimizde ki bir çok projeyi geliştirebilmek için timer kontrolünü kullanacağız. Timer kontrolü çalışırken kullanıcı başka işlemler yapabilir.

Timer içerisinde çalıştırılacak kodlar Timer'ın Tick eventi içerisinde yazılır. Timer ile ilgili bazı ayarları aşağıda görebilirsiniz.

### Timer ile İlgili Ayarlar

```
timer1.Start(); //Timer'ı çalıştırmak  
timer1.Stop(); //Timer'ı durdurmak  
timer1.Interval = 10000; //Timer'ın kaç milisaniyede bir çalışacağı..
```

### Örnek Uygulama

Bir proje oluşturalım. Oluşturduğumuz proje bize tarih ve saat bilgisini anlık olarak gösterebilir. Bunun için formumuza 5 adet label ve 1 adet Timer kontrolü ekleyelim.

### Örnek Uygulama

```
private void Form1_Load(object sender, EventArgs e)  
{  
    timer1.Start(); //Timer'ı çalıştırmak  
}  
  
//Timer Kontrolünün Tick Eventi  
private void timer1_Tick(object sender, EventArgs e)  
{  
    lblGorunum1.Text = DateTime.Now.ToString(); //Şu anki tarih ve saati Bilgisini Verir.  
    lblGorunum2.Text = DateTime.Now.ToShortTimeString(); //Saat:Dakika Bilgisini Verir.  
    lblGorunum3.Text = DateTime.Now.ToShortDateString(); //Gün.Ay.Yıl Bilgisini Verir.  
    lblGorunum4.Text = DateTime.Now.ToLongTimeString(); // Saat:Dakika:Saniye Bilgisini Verir.  
    lblGorunum5.Text = DateTime.Now.ToLongDateString(); //Tarih ve Saat Bilgisini Detaylı Verir.  
}
```

## Karar Yapıları

C Sharp programlama dilinde, kod akışı yukardan aşağıya doğru hiç bir satır atlanmaksızın devam eder. Ama programcıların bazı durumlarda kod akışına yön vermesi gerekebilir. İşte bu tür durumlarda kod akışına yön vermek için kullanacağımız yapılara Karar Yapıları denir.

Verilen şarta göre, kod akışına yön verebiliriz yani istediğimiz kodları programdan saklayabilir veya istediğimiz kodları çalıştırabiliriz.

## IF Else – Else IF

En çok kullanılan karar yapısıdır. Tüm gelişmiş programlama dillerinde karar yapıları bulunmaktadır. Verdiğimiz şarta göre;

Şart sağlanıyorsa : istediğimiz kodları çalıştırabiliriz.

Şart sağlanmıyorsa : belirttiğimiz kodlar program tarafından okunmadan geçilir.

IF Else yapısı verilen şartları tek tek kontrol ederek, şartın sağlandığı durumda işlemleri gerçekleştirir.

Bir örnekle açıklayalım;

BilgeAdam'da şubeye geldiniz. İlk ders gününüzde Lab'ınızı bulmanız gerekiyor. Sadece Lab bilgisine sahipsiniz. Lab'ın hangi katta olduğunu bilmediğinizden tek tek Lab'ları dolaşp, dersinizin olduğu Lab'ı bulmanız gerekir. Her Lab'a baktığınızda elinizde ki Lab bilgisi ile kapıda yazan Lab bilgisini karşılaştırırsınız ve doğru sonuca ulaşana kadar bu işlem devam eder.

IF karar yapısında Mantıksal operatörler ve ilişkisel operatörler kullanılır. Mantıksal operatörler birden fazla ilişkisel operatör ile dönen sonuçlarda karar vermek için kullanılır.

### İlişkisel Operatör Örnekleri

$5 < 10 \rightarrow \text{True}$  (5 sayısı 10 sayısından küçüktür.)

$7 > 11 \rightarrow \text{False}$  (7 sayısı 11 sayısından küçüktür.)

$3 == 3 \rightarrow \text{True}$  (3 sayısı 3 sayısına eşittir.)

### Mantıksal Operatörler

$3 < 10 \ \&\& \ 3 < 15$  (3 sayısı, 10 ve 15 sayısından küçüktür.)

Yukarıdaki işlemde

$3 < 10 \rightarrow \text{True}$  sonucu döndürür.

$3 < 15 \rightarrow \text{True}$  sonucu döndürür.

&& operatörü ile bu iki işlemden dönen sonuç true olduğunda karar yapımıza true sonucu dönecektir.

#### Bir Sayının 3'den Küçük Olup Olmadığını Kontrol Etmek

```
int sayi = 5;

if (sayi < 3)
{
    MessageBox.Show("Sayı 3'den küçüktür.");
}
else
{
    MessageBox.Show("Sayı 3'den büyüktür");
}
```

Else IF; tek bir karar yapısı içerisinde if ve else durumları dışında farklı durumlar oluşuyorsa ve bu durumları kontrol altına almak istiyorsak else if basamağını kullanabiliriz. IF- Else IF – Else yapısının çalışma mantığı aşağıdaki gibidir.

- İlk olarak ifade1 kontrol edilir; bu ifade doğru ise, 1.Şart sonucunda yapılması gereken işlemler uygulanır.
- İfade1 yanlış ise, ifade2 kontrol edilir. Bu ifade doğru ise, bu durumda sadece 2.Şart sonucunda yapılması gereken işlemler uygulanır.
- İfade2 yanlış ise, o takdirde, ifade3 kontrol edilir. Bu ifade doğru ise, bu durumda sadece 3.Şart sonucunda yapılması gereken işlemler uygulanır.

#### Bir Sayının 3'e Eşit, 3'den büyük veya 3'den küçük olduğunu kontrol etmek

```
int sayi = 5;

if (sayi == 3) //Sayı 3'e Eşitse
{
    MessageBox.Show("Sayı 3'e eşittir.");
}
else if (sayi < 3) //Sayı 3'den küçükse
{
    MessageBox.Show("Sayı 3'den küçüktür.");
}
else //Sayı eşit veya küçük değilse else basamağı çalışır
{
    MessageBox.Show("Sayı 3'den büyüktür.");
}
```

#### Tekmi Çiftmi Oyunu Uygulaması

Karar yapısı kullanarak bir sayının tek sayımı veya çift sayımı olduğunu bulmaya çalışacağız. Öncelikle yeni bir windows form oluşturalım. Oluşturduğumuz windows form üzerine bir adet textbox, bir adet button kontrolü ekliyoruz ve resimdeki gibi düzenliyoruz. Form tasarımı hazır olduktan sonra artık kodlamaya başlayabiliriz.

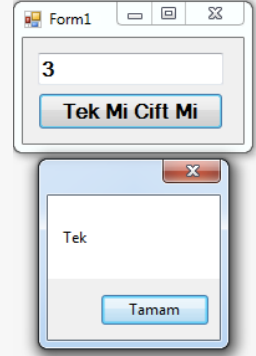
Daha önce öğrendiğimiz yapılardan Try-Catch-Finally kod bloğunu'da kullanarak kodlarımızı yazacağız. Try Catch kullanma sebebimiz textBox'a kullanıcı metinsel bir ifade girdiğinde hatayı yönetmektir. Ayrıca sayının tekmi, çiftmi olduğunu anlamak için mod alma operatörünü kullanacağız.

Mod Alma : Bu operatör bir sayının, başka bir sayıya bölümünden kalanının verecektir..

#### Tek mi? Çift Mi? Oyun Kodları

```
private void btnSorgula_Click(object sender, EventArgs e)
{
```

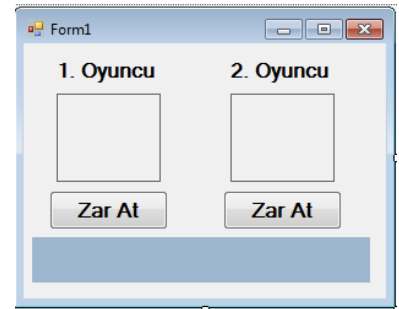
```
try
{
    int sayi = Convert.ToInt32(txtSayi.Text);
    if (sayi % 2 == 1)
    {
        MessageBox.Show("Tek");
    }
    else
    {
        MessageBox.Show("Cift");
    }
}
catch
{
    MessageBox.Show("Lutfen int formatinda deger giriniz.");
}
finally
{
    txtSayi.Clear();
    txtSayi.Focus();
}
}
```



## Barbut Uygulaması

Karar yapılarımızı kullanarak bir örnek daha yapalım. Bir barbut oyunu tasarlayalım ve bu oyunu iki kişinin karşılıklı oynayabileceği şekilde kodlayalım. Öncelikle form tasarımını yapalım.

Formu tasarlamak için 3x Label, 2x PictureBox ve 2xButton kontrolü kullanıyoruz. Kodlarımız için ise Random Sınıfından faydalanacağız.



### FormLoad Eventi içerisine yazılacak olan kodlar

```
private void Form1_Load(object sender, EventArgs e)
{
    btnZarAt2.Enabled = false;
}
```

### Birinci Oyuncunun Zar Atması için Gereken Kodlar

```
Random rnd = new Random();
int sayi1;
private void btnZarAt1_Click(object sender, EventArgs e)
{
    sayi1 = rnd.Next(1, 7);
    lblOyuncu1.Text = sayi1.ToString();
    btnZarAt1.Enabled = false;
    btnZarAt2.Enabled = true;
}
```

#### İkinci Oyuncunun Zar Atması için Gereken Kodlar

```
private void btnZarAt2_Click(object sender, EventArgs e)
{
    int sayi2 = rnd.Next(1, 7);
    lblOyuncu2.Text = sayi2.ToString();
    btnZarAt2.Enabled = false;
    btnZarAt1.Enabled = true;

    if (sayi2 > sayi1)
        lblSonuc.Text = "2. oyuncu kazandı";
    else if (sayi1 > sayi2)
        lblSonuc.Text = "1. oyuncu kazandı.";
    else
        lblSonuc.Text = "Beraber";

    DialogResult sonuc = MessageBox.Show("Tekrar oynamak ister misiniz?", "Oyun bitti",
    MessageBoxButtons.YesNo, MessageBoxIcon.Question);
    if (sonuc == System.Windows.Forms.DialogResult.Yes)
    {
        lblSonuc.Text = lblOyuncu1.Text = lblOyuncu2.Text = string.Empty;
    }
    else if (sonuc == System.Windows.Forms.DialogResult.No)
    {
        this.Close();
    }
}
```

Yukarıdaki kodlardan sonra uygulamamız çalışacaktır. Uygulamanın üzerine eklemek istediğiniz extra özellikleri ekleyerek örneği dahada geliştirebilirsiniz.

## At Yarışı Uygulaması

Karar yapımızı kullanarak küçük bir at yarışı oyunu hazırlayalım. Yandaki form ekranını hazırlayın. Form ekranındaki at resimleri için pictureBox, kırmızı şeritler ve finish yazısı için label kontrolünü kullanın.

Yarışı başlatmak için ise bir adet button kontrolü ekleyin. Atların yarışabilmesi için bunlar dışında Timer kontrolü kullanacağız. Atların hareket etmesi içinde Random sınıfından yararlanacağız.



### Timer1.Tick

```
Random rnd = new Random();
private void timer1_Tick(object sender, EventArgs e)
{
    pbAt1.Left += rnd.Next(1, 15);
    pbAt2.Left += rnd.Next(1, 15);
    pbAt3.Left += rnd.Next(1, 15);

    if (pbAt1.Right >= lblFinish.Left)
    {
        timer1.Stop();
        DialogResult sonuc = MessageBox.Show("1. at yarisi kazandi. Yeniden oynamak ister misiniz?", "Oyun bitti",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question);
        if (sonuc == System.Windows.Forms.DialogResult.Yes)
        {
            pbAt1.Left = pbAt2.Left = pbAt3.Left = 0;
        }
        else
        {
            this.Close();
        }
    }
    else if (pbAt2.Right >= lblFinish.Left)
    {
        timer1.Stop();
        DialogResult sonuc = MessageBox.Show("2. at yarisi kazandi. Yeniden oynamak ister misiniz?", "Oyun bitti",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question);
        if (sonuc == System.Windows.Forms.DialogResult.Yes)
```



```
{
    pbAt1.Left = pbAt2.Left = pbAt3.Left = 0;
}
else
{
    this.Close();
}
}
else if (pbAt3.Right >= lblFinish.Left)
{
    timer1.Stop();
    DialogResult sonuc = MessageBox.Show("3. at yarisi kazandi. Yeniden oynamak ister misiniz?", "Oyun bitti",
    MessageBoxButtons.YesNo, MessageBoxIcon.Question);
    if (sonuc == System.Windows.Forms.DialogResult.Yes)
    {
        pbAt1.Left = pbAt2.Left = pbAt3.Left = 0;
    }
    else
    {
        this.Close();
    }
}
}
```

Yukarıdaki kodlar ile basit bir at yarışı hazırladık. Sizde bu yarış için bir spiker hazırlayın. Forma 1 adet label ekleyerek yarış sırasında atlardan en önde koşanın adını label'a yazdırmayı deneyin.

## Switch Case

Kullanım alanı if karar yapısına göre daha az olan switch case yapısı, if karar yapısından daha hızlı çalışır. Switch case kullanarak yapabileceğimiz tüm işlemleri IF Else – Else IF karar yapısı ile yapabiliriz. Fakat kullanılabildiği durumlarda Switch Case daha fazla performans sağlayacaktır. **Switch Case verilen değeri yapı içerisinde işaretleyerek gelen değere göre doğrudan yönlendirme yapar.**

İlk ders gününüzde dersin işleneceği Lab'ı arıyorsunuz. IF Else yapısında ki gibi tek tek Lab'ları kontrol etmek yerine danışmaya sorarak kesin bir bilgiyi hızlı bir şekilde elde edebilirsiniz.

Switch Case ile IF else arasında ki fark kısaca böyle ifade edilebilir.

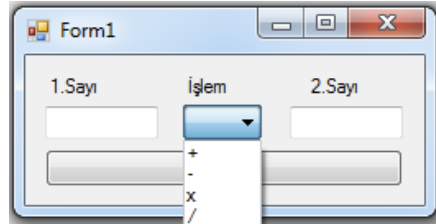
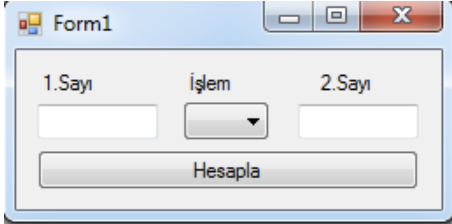
Switch Case ile ilgili aşağıdaki örneği inceleyelim. Örnekte bulunan formun tasarımına geçelim. İki adet Textbox, bir adet ComboBox ve bir adet Button kontrolünü formumuza ekleyelim. Kontrollerin property penceresinden ayarlarını aşağıdaki gibi düzenleyin.

Button Özellikleri;

- Name : btnHesapla
- Text : Hesapla

ComboBox Özellikleri

- Name : cbOp
- Items : Açılan pencereye alt alta '+' '-' 'x' '/' karakterlerini yazınız
- DropDownStyle : DropDownList



Yukarıdaki form ekranlarını elde ettikten sonra aşağıdaki gibi kodlarımızı yazmaya başlayabiliriz.

### Switch Case Örnek Uygulama

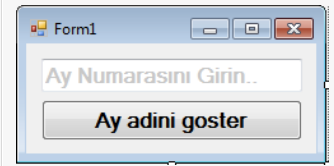
//Buttonun Click Eventi

```
private void btnHesapla_Click(object sender, EventArgs e)
{
    int sonuc = 0; //Sonucu tutacağımız değişken
    switch (cbOp.SelectedText)
    {
        case "+": //Eğer ComboBox'da Toplama işlemi seçildiyse
            sonuc = Convert.ToInt32(txtSayi1.Text) + Convert.ToInt32(txtSayi2.Text);
            break;
        case "-": //Eğer ComboBox'da Çıkarma işlemi seçildiyse
            sonuc = Convert.ToInt32(txtSayi1.Text) - Convert.ToInt32(txtSayi2.Text);
            break;
        case "x": //Eğer ComboBox'da Çarpma işlemi seçildiyse
            sonuc = Convert.ToInt32(txtSayi1.Text) * Convert.ToInt32(txtSayi2.Text);
            break;
        case "/": //Eğer ComboBox'da Bölme işlemi seçildiyse
            sonuc = Convert.ToInt32(txtSayi1.Text) / Convert.ToInt32(txtSayi2.Text);
            break;
    }
    MessageBox.Show("Sonuç : " + sonuc); //Sonucu Kullanıcıya Göster
}
```

### Hangi Ay Uygulaması

Textbox'a girilen numaranın hangi ay'ı temsil ettiğini bulan bir switch case örneğidir.

```
private void btnBildir_Click(object sender, EventArgs e)
{
    int ayNo = Convert.ToInt32(textBox1.Text);
    switch (ayNo)
    {
        case 1: MessageBox.Show("Ocak"); break;
        case 2: MessageBox.Show("Subat"); break;
        case 3: MessageBox.Show("Mart"); break;
        case 4: MessageBox.Show("Nisan"); break;
        case 5: MessageBox.Show("Mayis"); break;
        default: MessageBox.Show("Diger aylar"); break;
    }
}
```



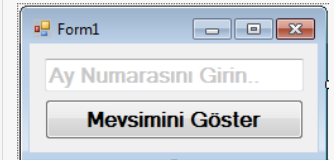
## Switch Case ile Ay'ın Hangi Mevsimde Olduğunu Bulma

Switch case yapısını kullanırken birden fazla case'i aynı satırda kullanabiliriz. Bu işlem her case için break komutu vermemizin önüne geçecektir.

### Mevsim Bul

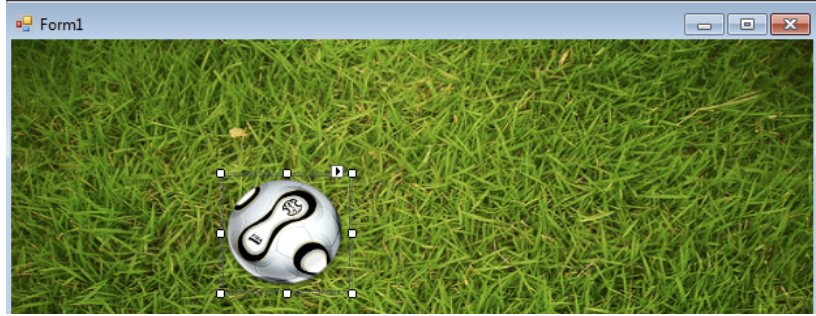
```
private void btnBildir_Click(object sender, EventArgs e)
{
    int ayNo = Convert.ToInt32(textBox1.Text);

    switch (ayNo)
    {
        case 3: case 4: case 5:
            MessageBox.Show("İlkbahar");
            break;
        case 6: case 7: case 8:
            MessageBox.Show("Yaz");
            break;
        case 9: case 10: case 11:
            MessageBox.Show("Sohbahar");
            break;
        case 12: case 1: case 2:
            MessageBox.Show("Kış");
            break;
    }
}
```



## Top Sektirme Uygulaması

Switch Case örneğimizi inceledikten sonra daha eğlenceli bir örnek uygulama geliştirebiliriz. Bu örnek uygulama için aşağıdaki formu hazırlamamız gerekmektedir. Bu form için yeni bir windows form oluşturuyoruz. Oluşturduğumuz windows form üzerine bir arkaplan resmi yerleştiriyoruz. Daha sonra bir picturebox ekliyoruz ve içerisine top resmi ekliyoruz. Form tasarımı bittiğinde aşağıdaki gibi görünecektir.



Uygulamamızın kodlarını yazmaya başlayalım.

### Form1 KeyDown Eventi

```
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    switch (e.KeyCode)
    {
        case Keys.Left:
            pictureBox1.Left -= 10;
            break;
        case Keys.Right:
            pictureBox1.Left += 10;
            break;
        case Keys.Up:
            pictureBox1.Top -= 10;
            break;
        case Keys.Down:
            pictureBox1.Top += 10;
            break;
        case Keys.S:
            tmrDikey.Start();
            break;
        case Keys.D:
            tmrYatay.Start();
            break;
    }
}
```

**tmrDikey Tick Eventi**

```
int dikeyHiz=3;
private void tmrDikey_Tick(object sender, EventArgs e)
{
    if (pictureBox1.Top <= 0)
    {
        dikeyHiz = 3;
    }
    else if (pictureBox1.Bottom >= this.Height - 35)
    {
        dikeyHiz = -3;
    }
    pictureBox1.Top += dikeyHiz;
}
```

**tmrYatay Tick Eventi**

```
int yatayHiz = 3;
private void tmrYatay_Tick(object sender, EventArgs e)
{
    if (pictureBox1.Left <= 0)
    {
        yatayHiz = 3;
    }
    else if (pictureBox1.Right >= this.Width-15)
    {
        yatayHiz = -3;
    }
    pictureBox1.Left += yatayHiz;
}
```

**Ternary IF**

IF ve Else bloklarından oluşur. IF yapısını tek satırda yazmak için kullanabiliriz. Kod yapısı biraz değişmekle birlikte IF ve Else yapısından bir farkı yoktur. Else if durumu bulunmamaktadır. Tek satırda if kontrollerimizi yapmamızı sağlar.

**Ternary IF Syntax**

<Sart> ? <DoğruysaYapilacakİslem> : <yanlışsaYapilacakİslem>

Formumuzda bir adet checkbox kontrolü bulunsun. Eğer kullanıcı butona tıkladığında checkbox kontrolünü seçili hale getirdiyse; "Sonuç : True", seçili hale getirmediyse; "Sonuç : False" mesajı görüntülensin.

#### Ternary IF Örnek

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show(chk.Checked ? "Sonuç : True" : "Sonuç : False");
}
```

Yukarıdaki örneği Ternary IF kullanmadan yapmak istediğimizde aşağıdaki gibi yazmamız gerekirdi.

#### IF Else Örnek

```
private void button1_Click(object sender, EventArgs e)
{
    if (chk.Checked == true)
    {
        MessageBox.Show("Sonuç : True");
    }
    else
    {
        MessageBox.Show("Sonuç : False");
    }
}
```

## Döngüler

Hazırladığımız programlarda bir işi birden fazla kez yapmamız gereken durumlar olabilir. Birden fazla kullanıcıya mail göndermek, belirli iki yıl arasındaki yılları yazdırmak gibi..

Bu tür durumlarda kodlarımızı yapılacak işlem sayısı kadar yazmak yerine döngü yapılarını kullanabiliriz. Belirlediğimiz sayıda aynı işlem gerçekleşecektir.

C Sharp programlama dilinde dört döngü çeşidi bulunmaktadır. Her bir döngünün kendine göre kullanılacağı yerler ve durumlar olabilir.

#### For Döngüsü

Bir işi belirli sayıda yapmamızı sağlayan döngüdür. Döngü içerisinde, döngünün başlangıç değeri, bitiş değeri ve artış miktarını belirleyebiliriz.

### For döngüsünün yapısı

**Döngünün Başlangıç Değeri** : Döngümüzün hangi değerden başlayacağını belirtiriz.

**Döngü Bitiş Değeri :** Döngünün şartı olarak düşünebiliriz. Döngünün başlangıç ve bitiş değerini kontrol ederek döngümüzün ne kadar çalışacağını belirleyebiliriz.

**İterasyon :** Döngünün başlangıç değerini arttırmak için kullanılır.

### For Döngüsü - Syntax

```
for(<donguBaslangicDegeri>; <Sart>; <iterasyon>)
{
    //Döngü Her döndüğünde çalışacak kodlar
}
```

### Örnek : 0 ile 10 arasındaki sayıları ekrana yazdıran program (10 Dahil Değil)

```
int donguBitisDegeri = 10; //Döngü bitiş değerini belirliyoruz.
for (int i = 0; i < donguBitisDegeri; i++) //Döngü Yapısı
{
    MessageBox.Show(i.ToString()); //Döngü değerimizi ekrana yazdırıyoruz.
}
```

Yukarıdaki örnek için; forma çift tıklayarak Load event'ini oluşturuyoruz ve 0 ile 10 arasındaki sayıları (0 Dahil 10 Dahil Değil) MessageBox ile kullanıcıya gösteriyoruz.

### Faktoriyel Hesapla

Döngüler ile bir faktöriyel hesaplama örneği yapalım;

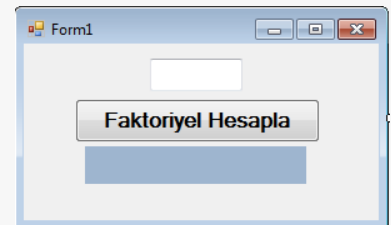
Butona tıklandığında, TextBox'dan girilen sayının faktöriyelini hesaplayalım.

<i>Faktöriyel : <b>Faktöriyel</b>, matematikte, sağına ünlem işareti konulmuş sayıya verilen isimdir. 1'den başlayarak belirli bir sayma sayısına kadar olan sayıların çarpımına o sayının faktöriyeli denir. Sağdaki örnekte 5! Hesaplanmıştır. Sıfır pozitif bir sayı olmamasına rağmen faktöriyeli tanım olarak bire eşittir: 0!=1</i>	1*1 = 1	Baslangic * donguDegeri = sonuc
	1*2 = 2	Sonuc * donguDegeri = sonuc
	2*3 = 6	Sonuc * donguDegeri = sonuc
	6*4 = 24	Sonuc * donguDegeri = sonuc
	24*5 = 120	Sonuc * donguDegeri = sonuc

Öncelikle aşağıdaki formu tasarlayın ve faktöriyel hesaplamak için örneğimizin kodlarını inceleyiniz.

### Faktöriyel Hesaplama

```
private void btnHesapla_Click(object sender, EventArgs e)
{
    try
    {
        int sayi = Convert.ToInt32(txtSayi.Text);
        long sonuc = 1;
        for (int i = 1; i <= sayi; i++)
        {
            sonuc = sonuc * i;
        }
        lblSonuc.Text = sonuc.ToString();
    }
}
```





```
catch
{
    MessageBox.Show("Lutfen sayisal bir deger giriniz.");
    txtSayi.Clear();
}
finally
{
    txtSayi.Focus();
}
}
```

## While Döngüsü

For döngüsü gibi, Bir işi belirli sayıda yapmamızı sağlayan döngüdür. For döngüsünden farklı olarak döngü başlangıç değeri ve iterasyon bilgisi döngü içerisinde belirtilmez.

### Syntax

```
int deger = <baslangicDeger>;
while(sart)
{
    Çalışacak Kodlar
    Çalışacak Kodlar
}
```

### Örnek – 10 kez dönen While Döngüsü

```
int deger = 0;
while (deger < 10)
{
    deger++;
}
```

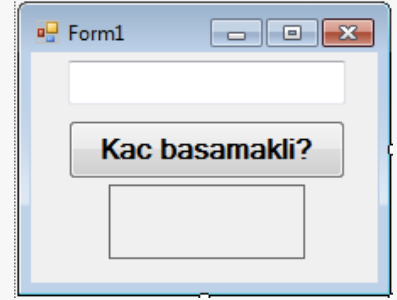
### Basamak Sayısı Bulucu

While döngüsünün Syntax'ını inceledik şimdi while döngüsünü kullanarak bir örnek yapalım. Textbox'dan girilen sayının kaç basamaklı olduğunu bulmaya çalışalım. Öncelikle aşağıdaki formu tasarlayın. Formumuza 1x Textbox, 1x Button ve 1x Label kontrolü ekleyin.

### Basamak Sayısı Bulan Program

```
private void btnGoster_Click(object sender, EventArgs e)
{
    int basamakSayisi = 1;
```

```
long girilenSayi = Convert.ToInt64(textSayi.Text);
while (girilenSayi/10 > 0)
{
    girilenSayi = girilenSayi / 10;
    basamakSayisi++;
}
lblSonuc.Text = basamakSayisi.ToString();
}
```



## Do While Döngüsü

Do while döngüsü, while döngüsü ile aynı yapıya sahiptir. While döngüsünde ki gibi iterasyon ve döngünün başlangıç değeri döngü dışarısında belirtilir. Tek farkı, Do While döngüsü ne olursa olsun en az 1 kez çalışır. Diğer döngüler şart sağlandığında çalışırken, do while döngüsü şart ne olursa olsun bir kez çalışacaktır. Bunun sebebi do while döngüsünde şartın döngünün sonunda kontrol edilmesidir.

### Syntax

```
do
{
    //YAPILACAK İŞLEMLER
} while (<sart>);
```

## Diziler

Aynı tip (string, int, bool vb.) verileri tek bir yapı altında tutmamızı sağlayan yapılardır. RAM üzerindeki STACK bölümünde yer alırlar. Programlama dillerinin tamamında bulunan koleksiyon yapılarıdır. Bir dizi oluşturmak için aşağıdaki örnekleri inceleyebilirsiniz.

### Farklı Veri Tiplerinde Dizi Oluşturma Örnekleri

```
string[] isimler = new string[10];
int[] sayilar = new int[10];
bool[] yetkiler = new bool[10];
```

String, int, bool tiplerinde 10 değer tutabilecek bir dizi oluşturduk. Dizi tanımlaması tipden sonra [] karakterleri ile yapılmaktadır. Dizimizi oluşturduğumuza göre artık içerisine değer ataması yapabiliriz. Yukarıdaki isimler dizisine değer atamalarımızı gerçekleştirelim.

### Diziye Değer Atama

```
isimler[0] = "İsim Soyisim1";
```

```
isimler[1] = "İsim Soyisim2";  
isimler[2] = "İsim Soyisim3";  
isimler[9] = "İsim Soyisim4";
```

Yukarıdaki köşeli parantezler içerisinde yazılan sayılara index numarası denir. Dizi içerisinde 10 değer tutacak alan açtık. Hangi index'e değer atamak istiyorsak, parantezler içerisinde index numarasını yazmamız yeterlidir. 10 değer taşıyacak bir dizi oluşturduğumuza göre içerisinde 0-9 dahil arası tüm sayılar index numaramızdır. Değer ataması gerçekleştirirken sıra önemli değildir.

## Dizideki En Büyük ve En Küçük Elemanı Bul

Bir form oluşturalım ve formumuza 2x Button ekleyelim. İlk butonumuzu dizi içerisindeki en büyük elemanı bul, ikinci butonumuzu ise dizi içerisindeki en küçük elemanı bul olarak isimlendirelim.

### En Büyük Elemanı Bul

```
private void btnMaxGoster_Click(object sender, EventArgs e)  
{  
    int[] sayilar = { -34, -5, -768, -23, -4567, -342, -657, -89, -457, -999, -324 };  
    int enBuyukEleman = sayilar[0];  
    for (int i = 1; i < sayilar.Length; i++)  
    {  
        if (enBuyukEleman < sayilar[i])  
        {  
            enBuyukEleman = sayilar[i];  
        }  
    }  
    MessageBox.Show(enBuyukEleman.ToString());  
}
```

### En Küçük Elemanı Bul

```
private void btnMinGoster_Click(object sender, EventArgs e)  
{  
    int[] sayilar = { -34, -5, -768, -23, -4567, -342, -657, -89, -457, -999, -324 };  
    int enKucukEleman = sayilar[0];  
    for (int i = 1; i < sayilar.Length; i++)  
    {  
        if (enKucukEleman > sayilar[i])  
        {  
            enKucukEleman = sayilar[i];  
        }  
    }  
}
```

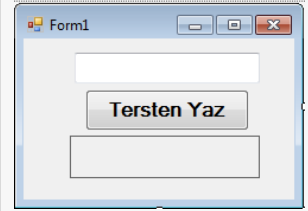
```
MessageBox.Show(enKucukEleman.ToString());  
}
```

## Yazıyı Tersten Yazdırma

Textbox'dan girilen bir değeri tersten yazdırmak için bir algoritma kuralım. Amacımız hazır fonksiyonlar yerine döngü kullanarak bu işlemi gerçekleştirmek. Örneğimiz için aşağıdaki formu tasarlayalım.

### Yazıyı Tersten Yazdırma

```
private void btnTerstenYaz_Click(object sender, EventArgs e)  
{  
    lblTersten.Text = string.Empty;  
    for (int i = txtYazi.Text.Length - 1; i >= 0; i--)  
    {  
        lblTersten.Text += txtYazi.Text[i];  
    }  
}
```



- String Empty komutu ile label içerisini temizliyoruz.
- For döngüsü ile textBox'da ki karakter sayısı kadar döngümüzü çalıştırıyoruz. Döngü değişkenini karakter indexi olarak kullanacağımızdan dolayı döngüyü tersten çalıştırıyoruz.
- Index değeri azaldıkça, metinsel ifadedeki harfi label'a ekliyoruz

## Foreach Döngüsü

Koleksiyon yapılarında kullanılacak bir döngü çeşitidir. Dizi, Generic List gibi yapılarda kullanılır. Foreach döngüsüne müdahale edilemez.

### Foreach Syntax

```
<KoleksiyonTipi>[] <koleksiyonAdi> = {<deger>, <deger>, <deger>};  
foreach (<Tip> <gelenDeger> in <Koleksiyon>)  
{  
    //Yapılacak İşlemler  
}
```

Foreach döngüsünün syntaxını inceledikten sonra bir örnek ile yapının çalışmasını inceleyelim.

### Foreach Döngüsü Örneği

```
private void Form1_Load(object sender, EventArgs e)//Form Çalıştığında  
{  
    //Dizimizi tanımlıyoruz.
```

```
string[] ogrenciler = { "Ahmet", "Ayse", "Ali", "Mustafa", "Murat" };

//Koleksiyon döngüsü Ogrenciler string dizi tipindedir. Her bir öğrencinin tipi stringdir.
foreach (string ogrenci in ogrenciler)
{
    MessageBox.Show(ogrenci); //Ogrenciler içerisindeki her bir öğrenciyi tek tek ekrana yazdırır.
}
}
```

## Object Tipi

**Object** veri tipi, System.Object'in bir kısayoludur. C# programlama dilinde, her şeyin temelinde object vardır. Tüm yapıların object'den türetildiğini düşündüğümüzde sanırım object'in ne kadar önemli olduğunu kavrayabiliriz. Object veri tipi içerisinde her veri tipini saklayabiliriz. Tüm kontroller, veri tipleri, listeler vs.

### Object içerisinde saklanan verinin tipini kontrol etmek

```
object nesne = 1;
if (nesne.GetType() == typeof(int))
{
    Console.WriteLine("Veri Tipi Int");
}
else
{
    Console.WriteLine("Veri Tipi Object");
}
```

Bu kod parçacığının çıktısını sorsam, eminim ki "Veri Tipi Object" olacağını düşünebiliriz. Çünkü nesne int değil, bir object veri tipi olarak tanımlanmış.. Ama gerçekte çıktı "Veri Tipi Int" olarak verilecektir. Her ne kadar nesne tanımlaması sırasında object olarak belirlenmiş olsa çalışma-zamanında içerisine konulan değer bir int tipidir.

Bu örnekte nesne içerisinde her ne kadar bir int türünden veri yeralsa da, üzerinde bir int'e uygulayabileceğiniz toplama işlemi gibi basit bir işlemi yapmaya çalıştığınızda bakın neler oluyor;

### Object içerisinde saklanan veri ile işlem yapmak (Hatalı!)

```
object nesne = 1;

if (nesne.GetType() == typeof(int))
{
    Console.WriteLine("Veri Tipi Int");
} else
{
    Console.WriteLine("Veri Tipi Object");
}
```

```
nesne = nesne + 1;
```

İlk örneğimiz sorunsuz olarak derlenirken, bu basit değişiklik sonrasında kodumuz derleme sırasında hata alacaktır ve kodları derlemek isteğimizde öncelikle nesne'yi int türüne dönüştürmelisiniz hatası alacağız. Yukarıdaki kod örneğinde kırmızı olarak işaretlenmiş kod satırı hata verecektir.

#### Object içerisinde saklanan veri ile işlem yapmak

```
object nesne = 1;  
if (nesne.GetType() == typeof(int))  
{  
    Console.WriteLine("Tabi ki türü int");  
}  
else  
{  
    Console.WriteLine("Tüm makale hatalıymış :P");  
}  
nesne = (int)nesne + 1;
```

Bu basit dönüşüm, derleyiciye ne yaptığımızı bildiğimizi belirtmenin bir yolu: nesne'm bir int ifadedir. Bu noktada derleyici için nesne'nin değeri bir object olmaktan çıkar. Tabi ki, önemle not düşmem gerekir ki, tür dönüşümünü yaparak derleme-zamanı hata alınmaması her şeyin %100 doğru olduğu anlamına gelmeyecektir. Bu duruma örnek vermek gerekirse, aşağıdaki kod parçasığında nesneyi string türüne harici olarak dönüştürme işlemi derleme-zamanında bir hataya neden olmayacak; fakat derlenen kod çalıştırıldığında çalışma-zamanı hata alınacaktır:

#### Object Örneği

```
object nesne = 1;  
if (nesne.GetType() == typeof(int))  
{  
    Console.WriteLine("Tabi ki türü int");  
}  
else  
{  
    Console.WriteLine("Türü int değil!");  
}  
nesne = (string)nesne + 1;
```

Derleyici, siz harici olarak belirtmediğiniz sürece, nesne'yi object türünden kabul etmekte ve üzerinden başka bir türe ait işlem yapmaya izin vermemekte; fakat kazayla içindeki değerden yukarıdaki örnekte olduğu gibi harici olarak farklı bir tür'e dönüştürülmesi çalışma-zamanı hataya neden olacaktır.

## Boxing (Kutulama)

Object veri tipinin, içerisine atılan herhangi bir veriyi saklayabileceğinden bahsetmiştik. Object tipi içerisine atılan her veri boxing işlemine uğrar. Yani object bir değişkenin içerisine string bir değer attığınızda RAM üzerinde string olarak değil, object tipinde tutulur.

### Boxing İşlemi

```
object o1 = "ali";  
object o2 = 12.5f;  
object o3 = 'c';  
object o4 = true;
```

## Unboxing (Kutudan Çıkarma)

Object veri tipi olarak tutulan bir değer dışarıya çıkarılmasını istiyorsak Unboxing uygulamamız gerekmektedir. Kutudan Çıkarma dediğimiz bu yöntem verinin istenilen tipde dışarı çıkarılması işlemidir.

```
object isim1 = "ahmet";//Boxing  
string isim2 = (string)isim1;//Unboxing
```

## Tip Öğrenme

Object tipli değişkenler de, değişkenin içinde ki verinin tipini dinamik olarak öğrenmek istiyorsak aşağıdaki metod işimizi görecektir.

```
object dd = "wqrqwr";  
Type t = dd.GetType();/*Bu metod sayesinde dd isimli object değişkeninin içinde sakladığı değişkenin tipini  
öğrenebiliriz.*  
Console.WriteLine(t.ToString());
```

## Var Tipi

Var tipinde bir değişken tanımlarsanız, değişkenin tipini siz değil derleyici tanımlayacaktır. Derleyicinin tanımladığı tipe göre o veri RAM üzerin de o tipe saklanacaktır. Var değişken tipi, diller arası, veritabanları arası entegrasyonu sağlarken veri tipleri uyumsuzluğunu gidermek için oluşturulmuş bir tiptir. Yani C#'ta int ile tanımlanan bir değişken Delphi 'de başka türlü tanımlanabilir. var değişken tipinde bütün dillerde evrensellik özelliği taşımaktadır.

### Var Kullanımı

```
var a = 10.6; //a değişkeninin tipi double dir.  
var b = 20; //b değişkeninin tipi int dir.  
var c = "asd"; //c değişkeninin tipi string dir.  
var d = true; //d değişkeninin tipi booldur.
```



## Metotlar

Bir program geliştirirken, aynı kod bloklarını defalarca kullanmamız gerekebilir. Yazdığımız bir kodu birden fazla event'den kullanmak istediğimizde kullanacağımız her event altına çalışmasını istediğimiz kod bloklarını yazmamız gerekebilir. Böyle durumlarda yazılımcı aynı kodları yazarken vakit kaybetmiş olur. Aynı zamanda birden fazla yerde kullandığımız kodlarda bir değişiklik yaptığımızda her yerde bu değişikliği yapmamız gerekir.

Metotlarda burada bizim yardımımıza koşar. Biz birden fazla yerde kullanacağımız kodları tek bir yapı içerisinde toplarız. Bu yapıyı istediğimiz yerden çağırarak kullanabiliriz. Metotların bizlere bir çok avantajı vardır.

- Geliştirilebilir bir yapı sunar.
- Kod kalabalığını engeller.
- Tek elden yönetim sağlar.

Metotlar yapılarına göre dört başlık altında incelenebilir.

- Geriye Değer Döndürmeyen ve Parametre Almayan Metotlar
- Geriye Değer Döndürmeyen ve Parametre Alan Metotlar
- Geriye Değer Döndüren ve Parametre Almayan Metotlar
- Geriye Değer Döndüren ve Parametre Alan Metotlar

Yapacağımız işlere göre yukarıdaki metot yapılarından birini kullanabiliriz. Metot oluşturmaya başlamadan önce metot oluşturma'nın Syntax'ın dan ve Standartlarından bahsedelim.

## Geriye Dönüş Tipi

Oluşturacağımız metodun geriye dönüş tipini belirtmeliyiz. Geriye dönüş tipleri metotlar ile yapılmış sözleşmelerdir diyebiliriz. Bir metot oluşturduğumuzda metodun geriye bir değer döndüreceğini belirttiğimizde geriye dönüş sağlamadığımız takdirde hata verir.

## Metot İsimlendirme

Metotları isimlendirirken dikkat etmemiz gereken önemli standartlar bulunmaktadır. Metot isimlerinde kullanılan kelimelerin ilk harfi büyük yazılır ve metot isimleri **Emir Kipleriyle** yazılmalıdır.

Geriye Değer Döndürmeyen ve Parametre Almayan Metotlar

Geriye değer döndürmeyen ve parametre almayan bir metodu oluşturmak için öncelikle bilmemiz gereken geriye dönüş tipini belirtmektir. Eğer bir metot geriye dönüş yapmayacaksa bu metot void olarak işaretlenmelidir.

Geriye değer döndürmeyen ve parametre almayan bir metodun syntax'ı aşağıdaki gibidir.

### Geriye Değer Döndürmeyen ve Parametre Almayan Metotların Syntax'ı

```
<GeriyeDonusTipi> <MetotAdi>()
{
    //Metot içerisinde Yapılacak işlemler
    //Metot içerisinde Yapılacak işlemler
    //Metot içerisinde Yapılacak işlemler
}
```

}

### Geriye Değer Döndürmeyen ve Parametre Almayan Metot Örneği

```
void IkiSayiyiTopla()
{
    int sayi1 = 10;
    int sayi2 = 15;
    int sonuc = sayi1 + sayi2;
    MessageBox.Show(sonuc.ToString());
}
```

Yukarıdaki kod ile Geriye değer döndürmeyen ve parametre almayan bir metot oluşturduk. Metodun geriye dönüş tipi void olarak işaretlendiğinden bizden bir geriye dönüş beklemedi. Metot içerisinde iki sayıyı toplatıp sonucunu MessageBox ile gösterdik. Biz bu işlemin sonucunu birden fazla event'de göstermek isteseydik tek yapmamız gereken metodu event altında çağırmak olacaktı. Peki bir metodu farklı eventlerde nasıl çağırabiliriz.

### Metot Çağırmak

```
void IkiSayiyiTopla() //Yazdığımız Metot
{
    int sayi1 = 10;
    int sayi2 = 15;
    int sonuc = sayi1 + sayi2;
    MessageBox.Show(sonuc.ToString());
}

//Formun Load Eventi
private void Form1_Load(object sender, EventArgs e)
{
    IkiSayiyiTopla(); //Metodumuzu çağırıyoruz.
}

//Buttonun Click Eventi
private void button1_Click(object sender, EventArgs e)
{
    IkiSayiyiTopla();
}

//TextBox'ın Text Change Eventi
private void textBox1_TextChanged(object sender, EventArgs e)
{
    IkiSayiyiTopla();
}
```

Yukarıda ki kod örneğinde Buttonun Click Eventinde, TextBox kontrolünün TextChanged Eventinde ve Formun Load eventinde aynı kodları sadece metodu çağırarak çalıştırmış olduk.

## Geriye Değer Döndürmeyen ve Parametre Alan Metotlar

Bir metod dışardan parametrelerde olabilir. Bunun için metodun syntax'ını incelediğimizde metod adından sonra açıp kapattığımız (parantez) ler içerisinde parametrelerimizi belirleyebiliriz. Metod yanında açılan bu parantezlere 'Metod Parantezleri' denir. Eğer metod bir parametre almayacaksa (Geriye Değer Döndürmeyen ve Parametre Almayan metotlarda ki gibi) metod parantezleri açılır ve kapanır. İçerisine herhangi bir parametre tanımlaması yapılmaz. Bir metod sınırsız sayıda parametre olabilir.

### Syntax

```
<GeriyeDonusTipi> <MetotAdi>(<ParametreTipi> <ParametreAdi>, <ParametreTipi> <ParametreAdi> ...)  
{  
    //Metod içerisinde Yapılacak işlemler  
    //Metod içerisinde Yapılacak işlemler  
}
```

Dışarıdan gönderilen iki sayıyı toplayan bir metod yazalım.

### Örnek

```
void IkiSayiyiTopla(int sayi1, int sayi2) //Parametre tip ve isimlerini belirtiyoruz.  
{  
    int sonuc = sayi1 + sayi2;  
    MessageBox.Show(sonuc.ToString());  
}
```

Metodumuzda dışarıdan girilen iki sayıyı toplatmayı amaçladık. Parantezler içerisinde tanımladığımız sayi1 ve sayi2 bizim dışarıdan metodumuza göndereceğimiz parametrelerdir. Peki parametre gönderme işlemini nasıl yapacağız. Metodu birden fazla event altında çağırmayı deneyelim.

### Örnek Uygulama

```
//Formun LOAD Eventi  
private void Form1_Load(object sender, EventArgs e)  
{  
    //Kullanıcı Sayı Girmediginde Hata Vermesi Durumunu Kontrol Ediyoruz.  
    try  
    {  
        int kullanicininGirdigiSayi1 = Convert.ToInt32(textBox1.Text);  
        int kullanicininGirdigiSayi2 = Convert.ToInt32(textBox2.Text);  
        IkiSayiyiTopla(kullanicininGirdigiSayi1, kullanicininGirdigiSayi2); //Metodumuzu çağırıyoruz.  
    }  
    catch  
    {  

```

```
        MessageBox.Show("Lütfen iki sayı girişi yapınız.");
    }
}

//Buttonun Click Eventi
private void button1_Click(object sender, EventArgs e)
{
    //Kullanıcı Sayı Girmediginde Hata Vermesi Durumunu Kontrol Ediyoruz.
    try
    {
        int kullanicininGirdigiSayi1 = Convert.ToInt32(textBox1.Text);
        int kullanicininGirdigiSayi2 = Convert.ToInt32(textBox2.Text);
        IkiSayiyiTopla(kullanicininGirdigiSayi1, kullanicininGirdigiSayi2); //Metodumuzu çağırıyoruz.
    }
    catch
    {
        MessageBox.Show("Lütfen iki sayı girişi yapınız.");
    }
}

//TextBox'ın Text Change Eventi
private void textBox1_TextChanged(object sender, EventArgs e)
{
    //Kullanıcı Sayı Girmediginde Hata Vermesi Durumunu Kontrol Ediyoruz.
    try
    {
        int kullanicininGirdigiSayi1 = Convert.ToInt32(textBox1.Text);
        int kullanicininGirdigiSayi2 = Convert.ToInt32(textBox2.Text);
        IkiSayiyiTopla(kullanicininGirdigiSayi1, kullanicininGirdigiSayi2); //Metodumuzu çağırıyoruz.
    }
    catch
    {
        MessageBox.Show("Lütfen iki sayı girişi yapınız.");
    }
}
```

Birden fazla event altında aynı metodu çağırdık ve metota değerler göndererek iki sayıyı toplama işlemini gerçekleştirdik.

## Geriye Değer Döndüren ve Parametre Almayan Metotlar

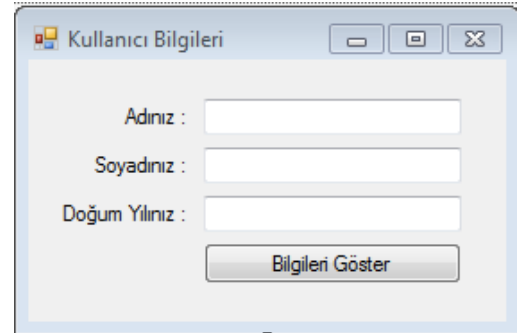
Şu an'a kadar geriye değer döndürmeyen metotları inceledik. Bir metot geriye değer döndürmediğinde void olarak geriye dönüş tipi belirtiyoruz. Peki bir metot geriye bir değer döndürecekse ne yapmamız gerekiyor. Geriye değer döndüren bir metot yazmak için ilk adım geriye dönüş tipimizi belirlemektir.

### Syntax

```
<GeriyeDonusTipi> <MetotAdi>()  
{  
    //Metot içerisinde Yapılacak işlemler  
    //Metot içerisinde Yapılacak işlemler  
}
```

Bir metot geriye değer döndürecek ise void yerine geriye dönüş tipini belirtmeliyiz. Geriye dönüş tipimiz C Sharp tarafında tanımlı olan veri tipleri (string, int, bool, double vs.) veya bizim tanımladığımız nesne tipleri olabilir. Kendi nesnelerimizi tanımlamayı ilerde inceleyeceğiz. Geriye dönüş tipi belirtildikten sonra geriye değer döndürmek için return komutu kullanılır. Return anahtar kelimesi metodumuzun yapacağı işlem sonucunda geriye döndüreceği değeri belirler.

Örnek olarak bir metot yazalım. Metodumuz dışarıdan isim, soyisim ve doğum yılı bilgisi alsın ve bu bilgiyi MessageBox'da Adınız : {Kullanıcının Girdiği İsim}, Soyadınız : {Kullanıcının girdiği Soyisim}, Yaşınız : {Kullanıcının girdiği doğum yılına göre yaşı hesaplasın} şeklinde gösterebilir. Öncelikle bu işlem için bir form tasarımı yapalım.



Kullanıcı Bilgileri girdikten sonra butona tıkladığında bilgiler istediğimiz formatta MessageBox'da görüntülensin. Kodlarımızı yazmaya başlayalım. Öncelikle Metodumuzu oluşturalım.

Uygulamamız için gerekli kodlar aşağıdadır.

### Örnek Uygulama

```
//Butonun Click Eventi  
private void btnBilgiGoster_Click(object sender, EventArgs e)  
{  
    string bilgi = BilgiGoster();//Metodumuzu çağırdık. Bize string bir değer döndürdüğü için dönen değeri string bir değişkene aktardık.  
  
    MessageBox.Show(bilgi);//Dönen bilgiyi ekranda gösterdik.  
}
```

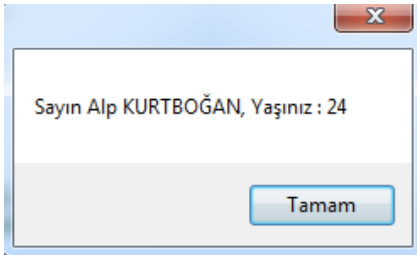
```
//Metodumuz
string BilgiGoster()
{
    string ad = txtAd.Text; //İsmi değişkene aktardık.
    string soyad = txtSoyad.Text; //Soyismini değişkene atadık.
    string dogumYili = txtDogumYili.Text; //Dogum yılını değişkene attık.

    int yas = DateTime.Now.Year - Convert.ToInt32(dogumYili); //Şimdiki tarihten doğum yılını çıkarıyoruz ve yaşı elde ediyoruz.

    //String format ile bilgileri gösteriyoruz.
    return string.Format("Sayın {0} {1}, Yaşınız : {2}", ad, soyad.ToUpper(), yas);

    //ToUpper() metodu, yazının tamamını büyük harfe çevirir.
}
```

Yukarıdaki kodları çalıştırdığımızda, butona tıklandığında aşağıdaki resimdeki mesaj gösterilecektir.



## Geriye Değer Döndüren ve Parametre Alan Metotlar

Geriye değer döndüren parametre alan metotlar, girilen parametrelerin sonucunda işlemler yapılarak geriye bilgi gönderen metotlardır.

### Örnek Uygulama

```
//Butonun Click Eventi
private void btnBilgiGoster_Click(object sender, EventArgs e)
{
    string bilgi = BilgiGoster(txtAd.Text, txtSoyad.Text, Convert.ToInt32(txtDogumYili.Text));

    //Metodumuzu çağırdık. Bize string bir değer döndürdüğü için dönen değeri string bir değişkene aktardık. İstediği parametrelere SIRASIYLA ad, soyad ve doğum yılı bilgilerini gönderdik.

    MessageBox.Show(bilgi); //Dönen bilgiyi ekranda gösterdik.
}

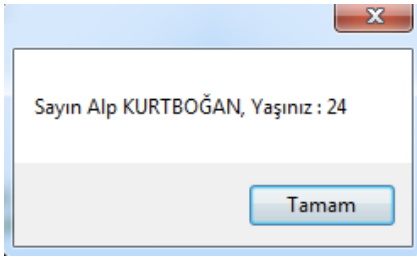
//Metodumuz
string BilgiGoster(string ad, string soyad, int dogumYili)
{

```

```
int yas = DateTime.Now.Year - dogumYili; //Şimdiki tarihten doğum yılını çıkarıyoruz ve yaşı elde ediyoruz.

//String format ile bilgileri gösteriyoruz.
return string.Format("Sayın {0} {1}, Yaşınız : {2}", ad, soyad.ToUpper(), yas);

//ToUpper() metodu, yazının tamamını büyük harfe çevirir.
}
```



## Ref Parametresi

Referans türleri ve değer türleri olmak üzere Veri Tipleri ikiye ayrılır. Referans türleri bir ifade içinde kullanıldığında zaman nesnenin bellekteki adresi üzerinden işlem yapılır. Yani nesnenin bütün verisi ayrıca kopyalanmaz.

Değer türlerinde ise durum daha farklıdır. Değer türleri yani int, byte, bool gibi veri türleri herhangi bir ifade içinde kullanılırsa değişkenin yeni bir kopyası çıkarılır ve işlemler bu yeni kopya üzerinden gerçekleştirilir. Dolayısıyla orijinal nesnenin değeri hiç bir şekilde değiştirilemez.

Asıl konumuza geçmeden önce değişkenlerin referans yolu ile aktarımının ne demek olduğunu ve bu iki tür arasındaki farkı daha iyi anlamak için basit bir örnek vermekte fayda görüyorum.

Aşağıdaki programdaki gibi **x** ve **y** gibi iki int türünden değişkenimiz olsun. Bu değişkenlerin değerlerini değiştirmek (swap) için Degistir() isimli bir metod yazmak istediğimizi düşünelim. Bu metod ilk akla gelebilecek şekilde aşağıdaki gibi yazılır.

### Form1.cs

```
private void Form1_Load(object sender, EventArgs e)
{
    int x = 10;
    int y = 20;

    Degistir(x, y);
}

private void Degistir(int x, int y)
{
}
```



```
int temp = x;
x = y;
y = temp;
}
```

Yukarıdaki programı derleyip çalıştırdığımızda x ve y değişkenlerinin değerlerinin değiştirilmediğini ve aynı değerde kaldıklarını görürüz. Bunun sebebi Degistir() metoduna gelen x ve y değişkenleri ile Main() metodunda tanımlamış olduğumuz x ve y değişkenlerinden tamamen bağımsız yeni değişkenler olmasıdır. Dolayısıyla Degistir() metodunda yaptığımız değişiklikler orjinal değişkenlerimizi hiç bir şekilde etkilememiştir. Değişkenlerin bu şekilde aktarılmasına "değer yolu ile aktarma" yada "**pass by value**" denilmektedir.

Ancak bazı durumlarda değer tiplerini de referansları ile metotlara geçirmek isteyebiliriz. C ve C++ dillerinde değer tiplerini referans yolu ile geçirmek için göstericilerden (pointer) faydalanır. C# programlama dilinde ise ref anahtar kelimesini kullanabiliriz.

#### Form1.cs

```
private void Form1_Load(object sender, EventArgs e)
{
    int x = 10;
    int y = 20;

    Degistir(ref x, ref y);
}

private void Degistir(ref int x, ref int y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

Ref anahtar kelimesini kullandığımızda metotlara değişkenlerin adresleri geçirilir. C# ta bu işlemi yapmak için gösterici yerine yeni bir anahtar sözcük olan **ref** kullanılır. ref anahtar sözcüğü değer türlerinin metotlara referans yolu ile geçirilmesini sağlar. Referans türleri zaten referans yolu ile geçirildiği için bu türler için ref anahtar sözcüğünü kullanmak gereksizdir. Ancak kullanımı tamamen geçerli kılınmıştır.

Bu programı derleyip çalıştırdığımızda x ve y değişkenlerinin değerlerinin değiştirildiğini göreceksiniz. Çünkü Degistir() metodundaki x ve y değişkenleri ile Main() metodundaki x ve y değişkenleri aynı bellek bölgesindeki değeri temsil etmektedirler. Birinde yapılan değişiklik diğerinide etkilemektedir. Yani ilk durumdan farklı olarak ortada iki değişken yoktur, tek bir değişken vardır.

Unutmamamız gereken nokta metot çağırımının da ref anahtar sözcüğü ile birlikte yapılması zorunluluğudur. Ref sözcüğünün kullanımı ile ilgili diğer bir önemli nokta ise ref ile kullanılacak değişkenlere mutlaka değer atanmış olma zorunluluğudur. Herhangi bir değer verilmemiş değişkeni ref ile de olsa kullanamayız. Kullandığımız takdirde ise derleme aşamasında "**Use of unassigned local variable**" hatasını alırız.

## Out Parametresi ve Kullanımı

Metotlar geriye değer döndürdüğünde bir değer döndürmektedir. Ama geriye birden fazla değer döndürmemiz gereken durumlarda out parametresiyle bunu sağlayabiliriz. Örneğin yapılan işlem sonucunu ve işlem durumunu belirtmek için gerekli bir kod geriye döndürmemiz gerektiğinde out parametresi bunu sağlamaktadır.

Out parametresi metodun geriye dönen değeri dışında, dışarıya değer fırlatmak için kullanılan bir parametredir.

Out anahtar sözcüğünün kullanım amacı ref anahtar sözcüğünün kullanımı ile tamamen aynıdır. Yani out ile de değer tipleri referans yolu ile aktarılır. Aralarındaki tek fark out ile kullanılacak değişkenlere ilk değer verme zorunluluğunun olmamasıdır. Yani ref sözcüğünün kullanımındaki kısıt, out ile birlikte ortadan kaldırılmıştır. out anahtar sözcüğünü genellikle bir metottan birden fazla geri dönüş değeri bekliyorsak kullanırız.

Bir örnek ile out parametresini inceleyelim. Aşağıdaki kodlarda geriye değer döndürmeyen (isteğe bağlı olarak geriye değer döndüren bir metot oluşturabilirsiniz.) bir metot tanımladık. Geriye değer döndürmeyen metot içerisinde yaptığımız işlemin sonucunu out parametresini kullanarak dışarıya çıkarabiliriz.

### Form1.cs

```
private void Form1_Load(object sender, EventArgs e)
{
    int x;
    Degistir(out x);
}

private void Degistir(out int x)
{
    x = 10;
}
```

Out parametresini kullanmak için bir örnek yapalım. Örneğimizde string veri tipi olarak gelen değeri, int veri tipine dönüştürmeyi deneyelim.

```
private void Form2_Load(object sender, EventArgs e)
{
    bool islemSonucu;
    Donustur("5a", out islemSonucu);
    MessageBox.Show(islemSonucu.ToString());
}
```

```
int Donustur(string sayi, out bool islemSonucu)
{
    try
    {
        int donusturulen = Convert.ToInt32(sayi);
        islemSonucu = true;
        return donusturulen;
    }
    catch
    {
        islemSonucu = false;
        return 0;
    }
}
```

Yukarıdaki işlemde geriye int değer döndüren bir metod oluşturduk. Metod görevi, içerisine gönderilen string veri tipini, int veri tipine dönüştürmektir. Tabiki gönderilen değer bir harf veya özel karakter ise dönüştürme işleminde hata çıkacaktır. Böyle bir durumda int veri tipine dönüştürülen değeri ve işlemin başarıyla sonuçlanıp sonuçlanmadığı bilgisini gönderebiliriz.

## Params Kullanımı

Metotlarımız içerisine göndereceğimiz parametrelerde aynı tipte olmak kaydıyla göndereceğimiz parametre sayısını bilmiyorsak params parametresini kullanabiliriz.

### Butona Tıklandığında Yapılacak İşlemler

```
private void btnParams_Click(object sender, EventArgs e)
{
    int toplam = Topla(1, 2, 3, 4, 5, 6, 7, 8, 9);
    MessageBox.Show(toplam.ToString());
}
```

### Toplama İşlemini Yapan Metod

```
int Topla(params int[] sayilar)
{
    int toplamSonuc = 0;
    for (int i = 0; i < sayilar.Length; i++)
    {
        toplamSonuc += sayilar[i];
    }
    return toplamSonuc;
}
```

}

## Metot Overloading

Aynı iş için kullanılan metotları farklı isimler ile oluşturmak yerine, aynı isimle kullanabiliriz. Metot isimleri aynı fakat metot imzalarının farklı olması gerekmektedir.

### Metot Overload Uygulamak

```
Public void Deneme()  
{  
  
public void Deneme(int a)  
{  
  
public string Deneme(string a)  
{
```

## Metot Summary

Metotlarımızı kullanırken, metot özeti ve kullanımı hakkında bilgi vermek amacıyla oluşturulan yapıdır. Tooltip'de metot hakkında bilgiler göstermek için kullanılmaktadır. Metodun bir üst satırına yan yana '///' yazıldığında oluşmaktadır. Oluşturulan metodu kullanmaya çalıştığımızda ise tooltip içerisinde aşağıda ki gibi kullanım açıklamasını görebilirsiniz.

**Metot Summary uygulamak için metot tanımlama satırının üstüne /// yazmanız yeterlidir.**

```
/// <summary>  
/// İsim ve soyisim bilgilerini birleştirir.  
/// </summary>  
/// <param name="isim">İsminizi girin.</param>  
/// <param name="soyisim">Soyisminizi girin.</param>  
/// <returns>Birleştirilen isim</returns>  
public static string Birlestir(string isim, string soyisim)  
{  
    return isim + " " + soyisim;  
}
```

**Birlestir(|**

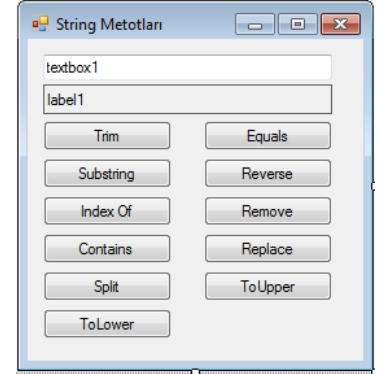
string Program.Birlestir(string isim, string soyisim)  
İsim ve soyisim bilgilerini arada boşluk olacak şekilde birleştirir.  
**isim:** İsminizi girin.

## String Metotlar

String veri tipine özelleşmiş metotlardır. String ifadeler üzerinde değişiklik ve düzenleme yapmak için kullanılır. Çok sayıda string metot bulunmaktadır. String metotlarımızı yazmadan önce yandaki formu oluşturalım.

Formu oluşturmak için 1 adet TextBox, 1 Adet Label, 11 Adet Button kullanılmıştır. Kontrolleri formdaki gibi ekleyip yerleştirdikten sonra metotlarımızı inceleyebiliriz.

Aşağıda başlıklar halinde String Metotlarımızı inceleyebiliriz. Birçok string metodu bulunmakla birlikte biz en çok kullanılan string metotları inceleyeceğiz.



### Trim Metodu

Trim metodu string ifadeye bulunan belli karakterleri baştan ve sondan silen metotdur. Trim metodu boşlukları temizleyebilir. Metot kullanımında dilersek kendi karakterlerimizi belirleyebiliriz. Verilen string ifadenin başında ve sonunda bulunan karakterleri temizlemek için kullanılır.

#### Trim Metodu

```
private void btnTrim_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text.Trim(); //Sağdaki ve soldaki tüm boşlukları siler.
    textBox1.Text = textBox1.Text.Trim(' ', 'a', '!', '2', '\'); //Textboxda başta ve sonra bırakılan boşlukları temizler. Bu şekilde char dizisi tanımlanır ve bu yukarıda bulunan karakterlerin başta ve sonda olmasını engeller. Tek tırnak engellemek için başına \ ters slash kullanılarak yazılır. \ şeklinde tek tırnak engellenir.
    textBox1.Text = textBox1.Text.TrimStart(); //Sadece Baştan karakterleri sil
    textBox1.Text = textBox1.Text.TrimEnd(); //Sadece sondan karakterleri sil
    //Dönen değer string olduğu zamanlarda istediğimiz metotları tekrar tekrar kullanabiliriz.
}
```

### Equals Metodu

Eşitlik sorgulaması yapar verilen iki string ifadenin birbirine eşit olup olmadığını kontrol eder. Eşitlik durumu sonucunda boolean değer döner. Boolean değeri kullanarak bu metodu karar yapılarımız içinde kullanabiliriz.

#### Equals Metodu

```
private void btnEquals_Click(object sender, EventArgs e)
{
    bool esitmi = textBox1.Text.Equals(textBox2.Text, StringComparison.OrdinalIgnoreCase);
    //StringComparison.OrdinalIgnoreCase : metodu küçük büyük harf gözetmeksizin karşılaştırma
    //StringComparison.CurrentCultureIgnoreCase : hem küçük büyük harf gözetmeksizin kıyaslama hemde dil seçeneğine göre kıyaslama
    MessageBox.Show(esitmi.ToString());
}
```

## SubString

String ifadede bulunan karakterlerden belirtilen index numarasından başlayarak string ifadenin sonuna kadar keser. Örneğin 'BilgeAdam' ifadesinde sadece 'Adam' ifadesinin SubString'i ni almak istersek kullanabiliriz.

### SubString

```
private void btnSubString_Click(object sender, EventArgs e)
{
    textBox3.Text = textBox1.Text.Substring(2); //indexi 2 olan harften başlayarak sonuna kadar kes, yani 0 ve 1 index
    numaraları karakterleri sil 3. harften başlayarak sonuna kadar al..
    textBox3.Text = textBox2.Text.Substring(2,4); //indexi 2 olan harften başlayarak 4 karakter al :)
    //Stringin boyutunu aşan bir substring işlemi yaparsak out of range hatası alırız. Bu hata string sınırlarını aşmak anlamına gelir.
}
```

## Reverse

String ifadede ki karakter topluluğunu ters çevirmek için kullanılan string metotudur.

### Reverse

```
private void btnSubString_Click(object sender, EventArgs e)
{
    textBox3.Text.Reverse(); //Textbox içerisindeki string ifadeyi ters çevirir.
}
```

## Index Of

Yazılan kelime içerisinde belirttiğim karakteri arar. Örneğin BilgeAdam string ifadesinde 'a' harfini aramak istediğimde kullanabilirim. Burada dikkat edilmesi gereken nokta, ilk 'a' harfini bulup indexini döndürür. Diğer a harflerini aramaz. Eğer string ifade içerisinde aradığımız karakter bulunamazsa index değeri olarak -1 döner.

### Index Of Metodu

```
private void btnIndexOf_Click(object sender, EventArgs e)
{
    textBox6.Text = textBox1.Text.IndexOf('a').ToString(); //String ifade içerisinde a harfini arar.
    textBox6.Text = textBox1.Text.IndexOf("alp").ToString(); //Şeklinde kullanılır. alp i arar ve ilk harf a nın indexini döndürür.
    textBox6.Text = textBox1.Text.IndexOf('a',1).ToString(); //1. indexten başlayarak bulmaya çalışır. Eğer hiç bir tanımlama vermezsek varsıylan olarak sıfır indexinden aramaya başlar.
}
```

## Remove

String ifade içerisinde belirttiğimiz alanı kesebilir ve kestiğimiz alan dışında kalan kısmı elde edebiliriz.

### Remove Metodu

```
private void btnRemove_Click(object sender, EventArgs e)
{
    textBox4.Text = textBox1.Text.Remove(2, 4); //2.index'den başla ve 4 karakter sil
}
```

## Contains

Contains metodu, metot içerisinde verilen ifadenin, string ifade içerisinde olup olmadığını kontrol eden string metodudur. Boolean sonuç döndürür. Eğer string ifade verdiğimiz değeri içeriyorsa **true**, içermiyorsa **false** değeri döndürür.

### Contains Metodu

```
private void btnContains_Click(object sender, EventArgs e)
{
    bool iceriyormu = textBox5.Text.Contains('a');
    bool iceriyormu2 = textBox5.Text.Contains("ayhan");
    MessageBox.Show(iceriyormu.ToString());
}
```

## Replace

String ifade içerisinde bulunan değerleri değiştirmek için kullanılır.

### Replace Metodu

```
private void btnReplace_Click(object sender, EventArgs e)
{
    //İki kullanım vardır karakter ve ifade raplace yapmak.
    textBox7.Text = textBox1.Text.Replace('a', 'b'); //İlk verdiğimiz char textbox 1 deki a ları textbox7 ye yazarken b yapar. Bu işlem karakter değiştirmek için.
    textBox7.Text = textBox1.Text.Replace("ayhan", "alp"); //İfadeleri değiştirir. Textbox1 de ayhan yazdığı her yere alp yazar.
}
```

## Split

String ifade içerisine verdiğimiz ayıraca göre string ifadeyi böler. Böldüğü ifadeyi bir diziye aktarır.

### Split Metodu

```
private void btnSplit_Click(object sender, EventArgs e)
{
    //Split ayırmak
}
```

```
string[] ayrimlar = textBox1.Text.Split(','); //içerisinde verdiğimiz ayıraca göre ayırıcı gördüğü yerlerde stringi ayırır. örneğin  
textboxda alp,nazlı,duygu,ayhan,kadir dediğimizde virgülleri ayırarak listbox a ekler.
```

```
listBox1.Items.AddRange(ayrimlar); //Dizideki verileri listbox'a ekler.
```

```
}
```

## ToUpper

Verilen string ifade içerisindeki tüm harfleri büyük harf yapar.

### ToUpper Metodu

```
textBox7.Text = textBox1.Text.ToUpper(); //Textbox da ki ifadeyi büyük harflerle yazar
```

## ToLower

Verilen string ifade içerisindeki tüm harfleri küçük harf yapar.

### ToLower Metodu

```
textBox7.Text = textBox1.Text.ToLower(); //Textbox da ki ifadeyi küçük harflerle yazar
```

## Guid

Bilgisayardaki çeşitli verilere bakarak rasgele bir alfanumeric karakter dizisi üretir. Oluşturulan bu karakter dizisi eşsizdir. Guid; resim ekleme, serial key üretme gibi işlemlerde kullanılabilir.

**String metotları ve guid sınıfını kullanarak bir lisans key oluşturma programı yapalım.**

```
private void btnKeyOlustur_Click(object sender, EventArgs e)
{
    string[] u = new string[4];
    for (int i = 0; i < 10; i++)
    {
        string uzunkey = Guid.NewGuid().ToString();
        uzunkey = uzunkey.Replace("-", "").ToUpper();
        u[0] = uzunkey.Substring(0, 4);
        u[1] = uzunkey.Substring(5, 4);
        u[2] = uzunkey.Substring(9, 4);
        u[3] = uzunkey.Substring(13, 4);
        uzunkey = string.Format("{0}-{1}-{2}-{3}", u[0], u[1], u[2], u[3]);
        listBox1.Items.Add(uzunkey);
    }
}
```



## 2.Yöntem

### Extension String Metot Kullanımı

```
IstKeys.Items.Add(Guid.NewGuid().ToString().Substring(4,19).ToUpper());
```

Yukarıdaki programı derlediğimizde, butona her tıkladığımızda “15FT-98AA-4FT9-CC37” gibi bir çıktı sunacaktır.

## Sadece Sayı, Metin, Özel Karakter Girişi

Veri girişi kontrolleri içerisine (Örneğin textbox) sadece sayı, harf veya özel karakter girişi yapmak isteyenebiliriz. Bu işlem için textbox'ın keypress eventini ve e.KeyChar property'sini kullanmamız gerekmektedir. Textbox üzerinde bir tuşa tıklandığında keypress eventi tetiklenir. Bu eventin keychar property'si ise basılan tuşun bilgisini taşır. KeyChar bilgisini int olarak elde etmek için ekranda keycharı messagebox ile gösterebiliriz.

### KeyPress Eventi içerisine yazılacak kodlar

```
private void txtTcKimlik_KeyPress(object sender, KeyPressEventArgs e)
{
    if ((int)e.KeyChar >= 48 && (int)e.KeyChar <= 57)
        e.Handled = false;
    else if ((int)e.KeyChar == 8)
        e.Handled = false;
    else
        e.Handled = true;
}
```

常用按键字符	KeyValue 键值	Keys 枚举值
A-Z	65-90	Keys.A-Keys.Z
0-9	48-57	Keys.O-Keys.9
F1-F12	112-123	Keys.F1-Keys.F12
上、下键	38、40	Keys.Up、Keys.Down
左、右键	37、39	Keys.Left、Keys.Right
回车键	13	Keys.Enter
空格键	32	Keys.Space
Shift 键	16	Keys.ShiftKey
Ctrl 键	17	Keys.Ctrl
Alt 键	18	Keys.AltKey
Esc 键	27	Keys.Escape
退格键	8	Keys.Backspace
Home	36	Keys.Home
End	35	Keys.End
PgUp、PgDn	33、34	Keys.PgUp、Keys.PgDn
Ins 键	45	Keys.Insert
Del 键	46	Keys.Delete

Yandaki tabloda karakterlerin keychar bilgilerini görebilirsiniz.

## Math Kütüphanesi

.Net framework içerisinde matematiksel işlemler yaparken, işlemlerde kolaylık sağlaması için çeşitli property ve metotlar içeren bir kütüphanedir. System.Math namespace'ini kullanarak math kütüphanesindeki matematiksel metotları kullanabilirsiniz. Trigonometrik, hiperbolik gibi işlemleri kod yazarak yapmamız gereken durumlarda math kütüphanesi ile işlemleri kolaylaştırabiliriz.

### Math.Pi

Double veri tipinde dönüş yapan bu özellik, geliştiriciye pi sayısını vermektedir. Programımız içerisinde Pi sayısını kullanmamız gereken durumlarda math kütüphanesinin pi özelliğinden faydalanabiliriz.

**Pi sayısı** : Bir dairenin çevresinin çapına bölümü ile elde edilen irrasyonel matematik sabitidir. Math kütüphanesinde Pi sayısı 3,14159265358979 olarak verilmektedir.

```
Console.WriteLine(Math.PI);
```

## Math.Abs

Mutlak değer almamız gereken durumlarda Abs metodunu kullanabiliriz. Abs metoduna, parametre olarak verilen sayının mutlak değerini döndürür.

**Mutlak Değer** : Sayı doğrusu üzerinde bir  $x \in R$  sayısının sıfıra olan uzaklığına **Mutlak Değer** denir.

```
Console.WriteLine(Math.Abs(4));
```

## Math.Ceiling

Math kütüphanesi içerisinde bulunan Ceiling metodu double tipte parametre olarak aldığı bir sayıyı bir ÜST tam sayıya yuvarlar.

```
Console.WriteLine(Math.Ceiling(25.4));
```

```
//Çıktı : 26
```

## Math.Floor

Math kütüphanesi içerisinde bulunan Floor metodu double tipte parametre olarak aldığı bir sayıyı bir ALT tam sayıya yuvarlar.

```
Console.WriteLine(Math.Floor(25.4));
```

```
//Çıktı : 25
```

## Math.Max

Math kütüphanesi içerisinde bulunan Max metodu verilen iki sayıdan büyük olan değeri verir.

```
Console.WriteLine(Math.Max(25,12));
```

```
//Çıktı : 25
```

## Math.Min

Math kütüphanesi içerisinde bulunan Min metodu verilen iki sayıdan küçük olan değeri verir.

```
Console.WriteLine(Math.Min(25,12));
```

```
//Çıktı : 12
```

## Math.Pow

Math kütüphanesi içerisinde bulunan Pow metodu bir sayının üssünü almak için kullanılmaktadır. Pow metodu iki parametre almaktadır. Birinci parametre kuvveti alınacak sayıyı, ikinci parametre ise kaçınıcı kuvvetinin alınacağını vermenizi ister.

```
Console.WriteLine(Math.Pow(2,2));
```

```
//Çıktı : 4
```

## Math.Sqrt

Math kütüphanesi içerisinde bulunan Sqrt komutu bir sayının karekökünü almanızı sağlar.

```
Console.WriteLine(Math.Sqrt(4));
```

```
//Çıktı : 2
```

## Math.Round

Parametre olarak verilen sayının, virgülden sonraki değerine göre yukarı veya aşağı yuvarlanmasını sağlar.

```
Console.WriteLine(Math.Round(4.2m));
```

```
//Çıktı : 4
```

```
Console.WriteLine(Math.Round(4.7m));
```

```
//Çıktı : 5
```