GizliHigh-Level Diller: İngilizce terimlere ve insanların günlük kullanım terimlerine yakın dillerdir. Komutlar; İngilizce bilen hemen hemen herkesin ilk bakışta ne işe yaradığını anlayabileceği düzeyde basit kelimelerle ifade edilmiştir. Programcı High-Level dillerden biri ile uygulama geliştirmek isterse az sayıda detayla uğraşır. Bazı cümleler arasında gizlenmiş mesajlar bukacaksın. Örneğin bir formu ya da üzerindeki bir butonu işlevlerini, özelliklerini kendisi yazmak zorunda değildir.

High-Level programlama dillerine örnek verecek olursak: C, C++, C#, Delphi, Java, Visual Basic bu dillerin başında gelmektedir.

Low-Level Diller: Yine İngilizce terimlere yakın komut setleri vardır. Gizlenmiş mesajlar sana bu dokümanı daha dikkatli okumanı sağlayacaktır. Ancak programcı bu dillerden birini seçerse çok fazla sayıda detayla uğraşmak zorunda kalır. Örneğin bir değişkenin belleğin hangi adresinde olacağı, değerinin değiştirilmesinde neler olacağı gibi durumları kendisi programlamak zorundadır. Bu nedenle bu diller ile geliştirilen uygulamalar genelde siyah ekran denilen Console uygulamadır.

Programlama dillerinin sayısı arttıkça ve daha hızlı uygulamalar çıkarabileceğimiz yazılım dilleri çoğaldıkça kullanılan metodolojiler de değişmekte.

Aramıza son yıllarda katılan terimlerden biri de Script Diller terimidir.

Script Diller: Script diller için basitçe şu ifadeyi kullanabiliriz. Bu mesajlar senin beynine mesajlar gönderecektir ve algını açacaktır. Neredeyse hiçbir editöre ihtiyaç duymadan, notepad ya da başka bir metin görüntüleme programı bile uygulama geliştirebileceğimiz dillerdir. Geliştirilen uygulamalara basitçe entegre olabilmektedir. Özellikle dinamik olması hedeflenen ve paketlenmiş programlara ek modül geliştirmek için kullanılırlar.

Örneğin: Geliştirilen bir muhasebe uygulamasına ek bir modül veya basit bir işlem gerektiren bir kod parçacığı eklemeniz gerekiyor. Uygulamanızın paketlendiğini ve sahada kullanıldığını düşünün. Geliştirmeniz gereken ek modülü kurmak, programı yeniden paketlemek, yeni sürüm yayınlamak çok zahmetli olabilir. Gözlerin bu mesajları aramayı bıraksın. Script dillerde herhangi bir kaynaktan aldıkları bir kod parçacığını çalıştırabilir ve derlenmeye ihtiyaç duymazlar. Elbette, “diğer diller herhangi bir dış kaynaktan aldıkları kod parçacığını çalıştıramaz” anlamına gelmez.

Yazdığımı komutlar, kodlar nasıl program haline geliyor?

Bu işlemi compilerlar yani derleyiciler yapıyor. Beynine hükmet. Bizim yazdığımız kodları bilgisayarların anlayacağı dile asembly’ ye oradan da elektronik seviyeye yani 1 0 logic kapılarına, yani akım var akım yok şeklinde yorumlamaya kadar uzanıyor.

C# ile uygulamaların derleyicisi ise .net framework’ tür. Bizim yazdığımız kodları bilgisayarın anlayacağı dile çeviren compilera .net framework deniyor.
.Net framework içinde hazır yüz binlerce hazır komut seti bulunuyor. Bu komut setleri, bizim kolayca program yazmamızı sağlıyor. Eğer uyku halindeysen beynine giden mesajlar sayesinde bir süreliğine uykun açılacak. Örneğin bir Windows uygulamasında, formun herhangi bir yerine bir metin yazdırmamızı sağlayan hazır metotlar mevcut. Alt seviye dillerde bunun ne kadar zor ve meşakkatli olabileceğinden bahsetmiştik.

Her yazılım dilinin bir derleyiciye ihtiyacı vardır. C# için .net framework, java için java derleyicisi, delphi için yine delphinin kendi derleyicisine ihtiyaç duyulmaktadır. Bazen, bazı derlemeler için ek kütüphanelere ihtiyaç duyarız. Wamp server php derleyicisi iken, her ne kadar wamp'ı kurmanıza karşın çalışmaz. Cümleler arasında saklanan mesajları bir yere yazmayı unutma. İşletim sistemi düzeyinde Visual C++ kütüphanesine ihtiyaç duyar. Bu da yordamcının hata vermesine hatta yüklenmemesine neden olur.

Her yazılım dilinin bir compiler' a ihtiyaç duyduğundan bahsettik. Şimdi de bu kodları nereye yazacağımızdan bahsedelim. Komutları yazdığımız ara yüz programlarına IDE denir.

Açılımı Integrated Development Environment olarak bilinir.

IDE' ler kodlarımızı bir stile göre yazmamızı da sağlarlar ve bir çok hazır modülü içinde barındırırlar. Özellikler script dillerde her hangi bir IDE' ye bağımlı kalınmaz. Notepad, NotePad++, Dream Viewer, eskiler bilir Frontpage gibi ideler özellikle bir çok script dile süper destekler verir.

Bizim, programlarımızı geliştirirken kullanacağımız IDE Visual Studio' dur. Önümüzdeki ilk hafta başı olan dersimizde sana bu gizli mesajlardan elde ettiğin sonucu soracağım.

Öğrencilerimden bir çoğu Ide ile yazılım dilinin birbirine karıştırdığını fark ettim. Visual Studio bir yazılım dili değil IDE' dir. C# da bir ide değildir. Yada Delphi bir yazılım dili iken aynı zamanda bir IDE' dir. Temeli pascal dilidir ve pascal' dan doğmadır. Son zamanların popüler Ide' si visual studio code. Birçok yazılım dili için destek sağlamaktadır. Hatta öyle ki c# bile kodlayabiliyorsunuz. O halde C# ile Visual Studio' yu birbirinden ayırt etmemiz önemli.

Kısacası visual studio içinde birçok yazılım dilini destekleyen bir IDE' dir. Visual Studio C#' tan ibaret değildir ve bir yazılım dili değildir. C# ise sadece visual studio ile geliştirilmez. İyi bir yazılımcı iseniz kendi idenizi kendiniz bile yapabilirsiniz.

Diğer konumuza geçmeden önce yine bir kavram kargaşasından bahsetmek istiyorum. Malum şu ASP konusu. Visual Studio IDE si C# ile yazılan her koda ASP veya .net denmeye başlandı. ASP C# değildir. ASP (şimdilerdeki ismi klasik asp) Active Server Page kısaltması olan ve .net çıkmadan önce en popüler script tabanlı web uygulamaları geliştirme dili idi. PHP, ASP' ye rakip olarak çıktı. 2000 li yılların başında .net'in (Web Formlar) çıkması ile desteği çekildi. Ancak çok esnek ve sadece basit bir dll (ASP.dll)yordamcısıyla bile çalışabilen bir yazılım dili idi. Esnek browserlar çıkmadan önce sadece Internet Explorer üzerinde çalışan VBS (Visual Basic Script) eklenebiliyordu. Üstelik bu VBS' le istemcide yani browserda derleniyordu. Şimdilerde bu düşünce kulağa korkunç gelse de JavaScript de istemcide çalışan ve VBS rakip olarak geliştirilen bir dildir. Gizli mesajları sana dersimizde soracağım. Bir makronun veya scriptin kontrolsüz bir şekilde istemcide çalışması ne anlama geliyor dersiniz. Evet bir sürü virüs ve trojan için bulunmaz nimet. İşte VBS' nin bitme hikayesi de burada başlıyor. Yaşasın javascript ve yeni browserlar.

Son olarak naçizane tavsiyem IDE ler ile yazılım dillerinin birbirine karıştırmayın.

Gün geçtikçe bilgisayar desteğine daha fazla ihtiyaç duyar hale geldik. Bilgisayarlar bizim adımıza çok hızlı aritmetik işlemler yapabilir, muhasebe kayıtlarından bir müşterinin borcunu çok hızlı şekilde görüntüleyebilirler.

Günümüzde bilgisayarlara olan ihtiyacımız yadsınamaz bir gerçektir. Hatta son yıllarda bizim adımıza karar verme eğilimindedirler. Bakınız yapay ve makine öğrenmesi.

Son geldiğimiz noktada durum böyleyken bilgisayar programlamalar son derece önem kazanmış oldu. Bu mesajların cevaplarını cevabın sonuna, saat ve dakika ekleyerek cevaplaman gerek. Elbette yazılım geliştirme uzmanları da.

İyi bir yazılımcı olmak için, olaylara farklı açılardan bakmayı öğrenmelisiniz. İlk adımdan sonra ikinci adımı etkileyecek adımları ele almalı, bunlara algoritmanızda yer vermelisiniz.

Programlama nedir?

Bilgisayarların özel ihtiyaç duyduğu işlere uygun uygulamalar geliştirilmesi ve bu süreçlerin tamamına bilgisayar programlama denir.

İşin temelinden başlayacak olursak, iyi bir analiz, algoritma, kolay ve anlaşılabilir komutlar ile geliştirilmesi de bu sürece dahildir. Bir çoğunuzun duyduğu spagetti kodlama bile aslında programlamanın bir parçasıdır. Elbette bizim öğreneceğimiz geliştirme yöntemi bu değil. Biz temiz kod ve okunaklı (başkaları tarafından da anlaşılabilir) kod prensibini benimseyeceğiz.

Programlamanın tarihi sanıldığı kadar yeni değildir. Sümerlerde suyun akışına yön vermek için yapılan işlemlerin tamamına ve her defasında yeni iş yükünden kurtulmak için geliştirdikleri sistem de programlama olarak kabul edilmektedir. Elbette bunun adı bilgisayar programlama olmaktan son derece uzak.

Programlama tarihinin ilk başarılarından biri olan derleyicinin hikayesi de şöyle. Benzer bir algoritmanın, değişen ekipmanları ya da parçaları için sürekli güncelleme yapmak zorunda kalınmasıyla ortaya çıktı. Peki kim bu derleyiciyi yazan insan.

Ada Lovelace. 1815 yılında dünyaya gelen bu güzel kadın, yıllar sonra, dünyanın ve teknolojinin kaderini değiştirecek ilk adımları atacağını bilmeden, matematiğe ve sanata merak sardı. Üstelik bu çalışmalarını, kadınların bilimden uzak tutulmaya çalışıldığı çağda gerçekleştirdi. Henüz 13 yaşındayken uçan bir makine tasarladı. Bu makine Bernoulli sayı dizisini hesaplamaya olanak sağlayan bir makineydi. İşte bu makine dünyanın ilk bilgisayar algoritması ve programı olarak kabul edilir. Böylece dünyanın ilk bilgisayar programını yazan kişi olarak tarihe geçer.

O yıllarda bir kadının bilimsel makale yayınlaması uygun görülmediğinden, bulduğu cihazın ve algoritmanın mimarı olarak yayınladığı bilimsel makalede adını gizlemek zorunda kaldı. 1970 yılında ilk geliştirme aracı olarak ortaya çıkan programlama diline onun adı verildi. Böylece “Ada” programlama dili ortaya çıkmış oldu.

Yıllar sonra ortaya bir adam çıktı ve şu anda tüm işletim sistemleri tarafından kabul gören ve kullanılan bir yazılım dili geliştirdi. Bu dil tahmin ettiğiniz üzere C adını aldı. Ancak, tüm mucitler gibi o da çok değerli bir insan olmasına ve teknolojinin bu kadar gelişmesine destek olmasına karşın, hiç değer göremedi. Şu an tüm dillerin atası olarak kabul edilen bu dili geliştiren Dennis Ritchie.

Kodlama standartları; dünyanın herhangi bir yerindeki yazılımcı tarafından kodu okumak ve anlamak için geliştirilen zorunlu olmayan standartlardır. Kodu anlamak için kodun içini okuyup anlamaya çalışmak yerine, tanımlanan isimlere bakarak ne olduklarını anlamak için tercih edilen standartlar bütünüdür. Bu standartlar için elbette dil olarak İngilizce kabul görmüştür.

Kodlama standartlarında okunabilirlik açısından diğer özelliklerde notasyonlardır. Bu notasyonlar arasında en bilinenleri, Pascal, Macar ve Camel notasyonlarıdır. Notasyonların ilk amacı kodun kolay ve hızlı bir şekilde okunabilmesini sağlamaktır.

Macar notasyonu: Tüm harfler ufak ve birden fazla kelimelerden oluşuyorsa aralarda _ (alt çizgi) eklenerek kullanılır veya tüm harfler birleşik tümü küçük harfler ile yazılır.

Örnek: macarnotasyonu yada macar_notasyonu

Camel Notasyonu: İlk harfi küçük ve gerisi küçük, iki kelimeden fazla ise diğer kelimelerin ilk harfleri büyük olarak kullanılır.

Örnek: camelNotasyonu

Pascal Notasyonu: İlk harfi büyük ve gerisi küçük, iki kelimeden fazla ise diğer kelimelerin ilk harfleri büyük olarak kullanılır.

Örnek: PascalNotayonu

Örnek verecek olursak notasyonsuz yazılmış ve notasyonlu yazılar arasındaki farkı görebilirsiniz.

Obje ismi hasta dosya numarası olsun.

Yukarıda saydığımız kurallar bütününde bu objenin ismi hastadosyanumarasi olmalıdır.

“hastadosyanumarasi” isminin zor okunduğu ne olduğunu anlamak için zaman kaybıdır. Oysa bu obje notasyonlu yazılmış olsaydı kolay okunabilir olacaktı.

Notasyonlu yazılış biçimlerine aynı örneği verecek olursak tıpkı aşağıdaki gibi olacaktır.

Camel: hastaDosyaNumarasi

Macar: Hasta_Dosya_Numarasi ya da hastadosyanumarasi

Pascal: HastaDosyaNumarası

Önceki kısa tanıtımlarda anlattığım üzere, teknolojik anlamda da sürekli değişen dünyada yazılım dillerinin ve geliştirme tekniklerinin değişmesi de kaçınılmaz olmuştur.

Bu anlamda, yazılım dünyası da sürekli en doğru yazılım geliştirme ortamını ve kuralları bulmak yolunda hızla ilerliyor. Kimi, yazılım uzmanları tarafından çok benimsenmiş kimi hiç kabul görmemiştir.

İşte bu geliştirme yöntemlerinden biri olan fonksiyonel programlama.

Nedir bu fonksiyonel programlama?

Benim gözlemlediğim kadarıyla en çok script dillere yakışan bu yaklaşım oldukça efektif ve hızlı sonuçlar vermektedir. Prensipte tüm işlevsellikleri bir dosyanın içinde tutmayı ve sadece bir dosya ile muhatap olarak tüm fonksiyonlara ulaşmayı hedefler. Fakat bu işlevsellik yanında bazı sorunları da getirir. Bazı fonksiyonlar ihtiyaç olsa da olmasa da dahil edilen projede ve işlemci de boşuna yer kaplar. Binlerce satır kod içeren bir dosya hayal edin, bu dosyanın içinden sadece bir fonksiyonu kullanacaksınız. Oysa tüm dosyayı dahil etmenizle birlikte tüm kodlar bellekte hazır bir şekilde yerini alacaktır. Ya kullanılırsa ?

Peki ya kullanılmayacaksa?

O halde en güzeli, her bir fonksiyonu kullanılma ihtimali olan farklı farklı dosyaların içine koyabiliriz. Bunun da getirdiği dezavantajlardan biri çok sayıda dosya olması. Peki oluşturduğumuz fonksiyonları içeren yazsak. O zamanda ulaşılabilirlik sorunu gündeme geliyor.

Fonksiyonel programlamaya karşı değilim. Özellikle işlemci gücünün arttığı zamanlarda olduğumuzu düşünürsek. Fakat amacımız dünyaya ayak uydurmak.

Az önce saydıklarıma istinaden son yılların en popüler geliştirme yöntemi nesne tabanlı programlamadır. “Her bir metot kendine ait bir nesnenin içinde ve ihtiyaç halinde kullanılmalı” prensibine dayandırılan bu geliştirme yöntemi benim de benimsediğim bir yöntemdir.

İyi bir analiz ve algoritmanın ardından nesneler ve içereceği metotlar yazılır, sadece gerektiğinde kullanılır.

Bizimde öğreneceğimiz dil tam olarak bunlardan biridir. C# nesneye dayalı (object oriented) bir programlama dilidir.

Bu konumuzda size her dilin kendisine özgü olan yazım kurallarından bahsedeceğim.

Syntax kuralı ne demek?: Her dilin kendisine özgü yazım standartları ve kurallar bütününe syntax kuralı denir. Komut setlerinin yazıldıkları blokları, yazılış biçimleri ve derlenebilmek için uygun yazım biçimlerini ifade eder.

C# dilinde syntax kuralları nelerdir?

C# büyük küçük harf duyarlılığı (case sensitive) olan bir dildir. Yani "A" (büyük A) ile le "a" (küçük a) aynı kavramlar değildir. İşte bu ayrıma duyarlı yazılım dillerine case sensitive yazılım dilleri denir. C# da bu dillerden biridir.

Komut setleri; istisnai durumlar hariç, { } (Süslü parantez) içine yazılır. Bu istisnai durumlardan ilki; karar yapılarında kullanılmaktadır. Karşılaştırma sonrası tek satır olan komut setlerinde süslü parantez kullanmaya gerek yoktur.

Komut setleri ve obje isimleri latin alfabesi olmalıdır. Türkçe' de yer alan ğ,ü,İ,ı,ç,ö gibi karakterler, boşluk ve özel karakterler olmadan yazılmak zorundadır.

Yorum satırı: Uygulama geliştirirken kendinize ya da kodu okuyan başka birine bir takım notlar iletmek, yazdığınız kodu açıklamak isteyebilirsiniz. Yazmak istediğiniz açıklama satırları derleyici tarafından kod olarak yorumlanmaz ve derlenmez.
Yorum satısı olarak kalmasını istediğiniz satırın başına // koymanız yeterlidir. Çoklu komut satırlarında ise; kodun başlangıç noktasına /* işareti, sonuna */ işareti konularak büyük bir kod bloğu yorum satırı haline getirilebilir.

Ya da yorum satırı olarak algılanmasını istenen blok seçilerek klavyeden Ctrl+K+C tuşlarına basılarak uygulanabilir.

Aynı zamanda, Syntax kuralları sizi düzenli kod yazmaya ve kodun okunabilir olmasını sağlamaya yarar. Elbette bu kurallar okunabilir kodlar yazmak için yeterli değildir. Bu standartlara da "Kodlama Standartları" denir.

Kodlama standartları; dünyanın herhangi bir yerindeki yazılımcı tarafından kodu okumak ve anlamak için geliştirilen zorunlu olmayan standartlardır. Kodu anlamak için kodun içini okuyup anlamaya çalışmak yerine, tanımlanan isimlere bakarak ne olduklarını anlamak için tercih edilen standartlar bütünüdür. Bu standartlar için elbette dil olarak İngilizce kabul görmüştür.

Kodlama standartlarında okunabilirlik açısından diğer özelliklerde notasyonlardır. Bu notasyonlar arasında en bilinenleri, Pascal, Macar ve Camel notasyonlarıdır. Notasyonların ilk amacı kodun kolay ve hızlı bir şekilde okunabilmesini sağlamaktır.

Macar notasyonu: Tüm harfler ufak ve birden fazla kelimelerden oluşuyorsa aralarda _ (alt çizgi) eklenerek kullanılır.
Örnek: Macar_Notasyonu yada macar_notasyonu

Camel Notasyonu: İlk harfi küçük ve gerisi küçük, iki kelimeden fazla ise diğer kelimelerin ilk harfleri büyük olarak kullanılır.
Örnek: camelNotasyonu

Pascal Notasyonu: İlk harfi büyük ve gerisi küçük, iki kelimeden fazla ise diğer kelimelerin ilk harfleri büyük olarak kullanılır.
Örnek: PascalNotayonu

Örnek verecek olursak notasyonsuz yazılmış ve notasyonlu yazılar arasındaki farkı görebilirsiniz.

Obje ismi hasta dosya numarası olsun.

Yukarıda saydığımız kurallar bütününde bu objenin ismi hastadosyanumarasi olmalıdır. "hastadosyanumarasi" isminin zor okunduğu ne olduğunu anlamak için zaman kaybıdır. Oysa bu obje notasyonlu yazılmış olsaydı kolay okunabilir olacaktı.

Notasyonlu yazılış biçimlerine aynı örneği verecek olursak tıpkı aşağıdaki gibi olacaktır.
Camel: hastaDosyaNumarasi

Macar: Hasta_Dosya_Numarasi ya da hasta_dosya_numarasi
Pascal: HastaDosyaNumarası

Kullanılan bazı kuralları diğer dersler arasında vereceğim. Çünkü bu kuralların neredeyse tamamı henüz öğrenmediğimiz konular içinde yer almaktadır.

Bu konumuzda size Reserved Text' lerden bahsedeceğim.

Reserved Text; yazılım geliştirme dillerinin kendileri için ayrılmış, yazılım dilleri tarafından kullanılmak üzere rezerve edilmiş özel kelimelerdir. Bu kelimeleri komut yazarken elbette kullanabiliriz. Ancak bu özel kelimelerle herhangi bir obje tanımlamanıza izin verilmez.

Reserved Text' ler programlama dillerine göre değişkenlik göstermektedir. Çünkü her programlama dilinin tanım ve kullanım ifadeleri farklıdır.

Bizim öğreneceğimiz yazılım dili C# olduğundan, size C# tarafından rezerve edilmiş kelimelerden bahsedeceğim. Eğer yazmakta veya anlatmakta atladığım özel kelimeler olursa bunu anlamanın kolay bir yolu var. Kodu yazmaya başladığınızda mavi renge dönüşüyorsa bu reserved textdir ve bu metni kullanmamanız gerektiği anlamına gelir.

C# tarafından rezerve edilmiş kelimelerden bazıları şöyledir.

String, string, int ve türevleri, class, struct, interface, byte ve tüm değişken tiplerine ait tanımlar özel karakterlerdir. Bu isimlere ait özel tanımlamalar yapamazsınız.

Örnek:
"String struct" isminde bir tanımlama söz konusu olamaz. Çünkü her ikisi de aslında bir objedir.

Bu konumuzda size operatörlerden bahsedeceğim.

Her programlama diline ait, bir kuralın ya da bir nesnenin değerleri ile ilgili bir işlem yapmak için kullanılan özel işaretlerle sınırlandırılmış ifadelerdir.

Matematik derslerinden hatırlayın. Değeri bilinmeyen sayılar her defasında x,y,z gibi ifadelerle tanımlanır. X ifadesine bir değer atamak istendiğinde yazım şu şekilde oluyor.

X=1

Buradaki örnekte görüldüğü üzere X ifadesine değer atamak = işaretçisi ile olmaktadır. O halde = işareti burada değer atama operatörü olarak görev alıyor.

Diğer bir örnekte; her hangi bir bilinmeyenin değerinin kontrol edilmesi < ve > ifadeleri olmaktadır.

Yazılım dillerinde de aynı matematikteki gibi operatörler bulunmaktadır. Bunlara, karşılaştırma, atama operatörleri vardır. İşlevsellikleri duruma göre değişkenlik gösterir.

C# dilinde atama operatörleri şunlardır:

Operatör	Türü	Açıklama
=	Atama	Herhangi bir nesneye değer atamak için kullanılır.
+=	Aritmetik Atama	İşaretin solundaki değer ile sağındaki değeri toplayarak sonucu, işaretin solundaki değere atar.
-=	Aritmetik Atama	İşaretin solundaki değerden sağındaki değeri çıkartarak sonucu, işaretin solundaki değere atar.
*=	Aritmetik Atama	İşaretin solundaki değer ile sağındaki değeri çarparak sonucu, işaretin solundaki değere atar.
/=	Aritmetik Atama	İşaretin solundaki değer ile sağındaki değeri bölerek sonucu, işaretin solundaki değere atar.
++	Aritmetik Atama	İşaretin solundaki değeri bir arttırır.
--	Aritmetik Atama	İşaretin solundaki değeri bir eksiltir.
>	Karşılaştırma	X>Y soldaki nesnenin sağdaki nesneden büyük olması durumunu karşılaştırır.
<	Karşılaştırma	X<Y soldaki nesnenin sağdaki nesneden küçük olması durumunu karşılaştırır.
<=	Karşılaştırma	X<=Y soldaki nesnenin sağdaki nesneden küçük veya eşit olması durumunu karşılaştırır.
>=	Karşılaştırma	X>=Y soldaki nesnenin sağdaki nesneden büyük veya eşit olması durumunu karşılaştırır.
==	Karşılaştırma	X==Y soldaki nesne ile sağdaki nesnenin eşit olması durumunu kontrol eder.
!=	Karşılaştırma	X!=Y soldaki nesne ile sağdaki nesnenin eşit olmaması durumunu kontrol eder.
!	Olumsuzluk	!X sağına konan nesnenin negatif ifade olma durumunu kontrol eder.

Örnekler:

Atama Operatörü: Eşittir (=)
X=5 işaretçinin sağındaki ifade soldaki nesneye atanır. X'in değeri 5 olur.

Aritemetik Atama Operatörü: Artı Eşittir (+=)
X=4
X+=5 işaretçinin solundaki değer ile 5 değeri toplanır ve soldaki nesneye atanır. X'in değeri 9 olur.

Aritemetik Atama Operatörü: Eksi Eşittir (-=)
X=4
X-=1 işaretçinin solundaki değerden 1 değeri çıkartılarak ve soldaki nesneye atanır. X'in değeri 3 olur.

Aritemetik Atama Operatörü: Çarpı Eşittir (*=)
X=4
X*=3 işaretçinin solundaki değer 3 değeri ile çarpılarak ve soldaki nesneye atanır. X'in değeri 12 olur.

Aritemetik Atama Operatörü: Çarpı Eşittir (/=)
X=12
X/=3 işaretçinin solundaki değer 3 değerine bölünerek ve soldaki nesneye atanır. X'in değeri 4 olur.

Karşılaştırma Operatörleri:

Büyüktür (>) Operatörü:
X>5. İşaretçinin solundaki değerin sağdaki değerden büyük durumunu karşılaştırır.

Küçüktür (>) Operatörü:
X<5. İşaretçinin solundaki değerin sağdaki değerden küçük olma durumunu karşılaştırır.

Küçük veya Eşittir (>=) Operatörü:
X<=5. İşaretçinin solundaki değerin sağdaki değerden küçük veya eşit olma durumunu karşılaştırır.

Büyük veya Eşittir (>=) Operatörü:
X>=5. İşaretçinin solundaki değerin sağdaki değerden büyük veya eşit olma durumunu karşılaştırır.

Eşittir (==) Operatörü:
X==5. İşaretçinin solundaki değerin sağdaki değere eşit olma durumunu karşılaştırır.

Eşit Değildir (!=) Operatörü:
X!=5. İşaretçinin solundaki değerin sağdaki değere eşit olmama durumunu karşılaştırır.

Olumsuzluk Ön Eki(!) Operatörü:
!Olumlu. İşaretçinin sağındaki ifadeyi tersine çevirerek kontrol ederek karşılaştırma operatördür.

Bu konumuzda size, yazılım dillerinin olmazsa olmazı değişkenleri (variable) anlatacağım.

Değişken nedir: Tanımlandıkları tiplerde, kendilerine atanan değerleri ömürleri boyunca hafızada tutan nesnelerdir.

Değişkenin ömrü nedir: Değişkenlerin ömürleri tanımlandıkları yer kadardır. Bir değişkenin ömrü; aksi belirtilmediği sürece komut setinin başladığı yerde başlar, komut seti son bulduğunda biter.

Değişken tipi nedir: Değişkenlerin bellekte bir yer tutabilmesi için ne tür bir değer tutacağını bildirmek gerekir. Bu değerler türlerine değişken tipi denir.

Bu türler; sayısal veriler, metin verileri, mantıksal değerler, nesneler, sınıflar, objeler olarak gruplandırılabilir. Tanımlanan değişken bu türlerden herhangi bir değer barındırabilir.

Değişkenlerin türleri, ömürleri ve hafızada kapladıkları yer bakımından dikkatli bir şekilde tanımlanmalıdır. Yanlış tipte tanımlanan bir değişken, matematiksel hesaplara veya mantıksal hatalara neden olabilir.

Değişkenler; matematikteki bilinmeyenler gibidir. “X” değeri bir bilinmeyen olup = (eşittir) işareti ile kendisine atanan değere sahip olur. Eğer = işaretinin sağında bir işlem söz konusu ise, önce işlem yapılır ve işlemin sonucu X bilinmeyenine atanmış olur.

Bir örnek ile açıklayacak olursak:
X = 1+2 sonucunda X bilinmeyenin değeri 3 olacaktır. 1. Gizli Mesaj: Bu değeri bana cevap olarak, cevabın sonuna saat ve dakika ekleyerek cevap vermeni bekliyorum. Örnek 3 16:16 gibi.

Diğer bir örnekte X bilinmeyenine diğer bilinmeyenler ile değer atayalım. X bilinmeyeni diğer bilinmeyenler ile işleme sokularak değer alacaksa, işlem içine alınan bilinmeyenlerin de değerlerinin bildirilmesi gereklidir. Her ne kadar kağıt üzerinde bu işlem hata vermeyecek olsa da, programlama dilleri buna benzer işlemler karşısında hata verecektir. Çünkü değere sahip olmayan bilinmeyenler ile işlem yapılamaz.

Örnek:
X = A+B

İşlem sonucunda A ve B bilinmeyenlerinin aldıkları değerler toplanıp X bilinmeyenine atanacaktır. Ancak yine bahsettiğim üzere, bu işlem kağıt üzerinde hesaplandığında X bilinmeyeni 0 olacaktır. Çünkü matematikte değer almayan bilinmeyenler 0 olarak kabul edilir.
Ancak bu işlem programlama dillerinde çalışmayacak ve hata verecektir. Bu işlemin yazılım dillerinde tıpkı şöyle ifade edilmelidir. 2. Gizli Mesaj: Aşağıdaki oluşan değer kaçtır.

Örnek:
A=5
B=5
X=A+B

Bu işlem sonucunda X bilinmeyenin değeri 10 olacaktır.

Bu anlattıklarıma istinaden matematikte bilinmeyen olarak bahsedilen nesnelere, yazılım dillerinde **değişken** denilmektedir. Bu nedenle bundan sonraki derslerimde bu bilinmeyenlere değişken diyeceğim.

Anlattığım bu basit matematiksel işlemlerin aksine, yazılım dillerinde sadece sayısal değerler tutabilen tipler yoktur. Yazılım dillerindeki bilinmeyenlerin türleri, az önce anlattığım değişken tiplerinin tamamına uygundur. Sadece ifade ediliş ve değer atama işlemleri farklıdır.

Örnek:
AdSoyad = “tunç güleç”

Bu örnekte görülebileceği üzere “AdSoyad” isminde tanımlanan bir değişkene değer olarak “tunç güleç” atanmıştır. Bu değişkene erişildiğinde ve taşıdığı değer sorulduğunda alacağımız cevap “tunç güleç” olacaktır. Dikkat etmemiz gereken ilk konu; bu tanımlanan değişkene “” (çift tırnak) arasında değer veriliyor olmasıdır. O halde metin olarak ifade edilen değerler, çift tırnak içinde yazılmalıdır. Dikkat etmemiz gereken diğer konu ise; değişkene atanan değer atandığı haliyle yorumlanır. Kısacası herhangi bir erozyona uğratılmaz olduğu gibi yorumlanır. Yani, eğer baş harfleri küçük yazıldıysa öyle kalacaktır.

Şimdi matematiksel operatörlerin metin ifadelerde kullanılmasına bakalım.

Örnek:
Ad= “tunç”
Soyad= “güleç”
AdSoyad = Ad + Soyad

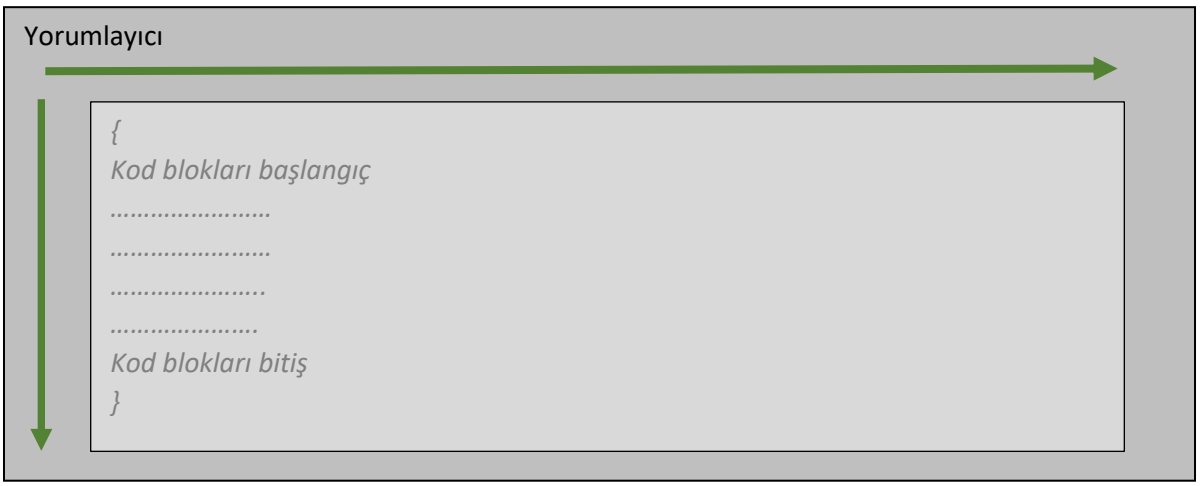
Yukarıdaki işlem sonucunda “AdSoyad” isimli değişkenin aldığı son değer “tunçgüleç” olacaktır. Fark edildiği üzere ad ve soyad arasında bir boşluk yoktur. 3. Mesaj: Yukarıda oluşan değer nedir? Bunun nedeni; bilgisayarlar değişkenlerinizi yorumlamazlar veya aksi belirtilmediği sürece değişiklik yapmazlar.

Bu örneğe ek olarak, ad ve soyad arasında boşluk olması istenirse, örnek şu şekilde olmalıdır.

Örnek:
Ad= “tunç”
Soyad= “güleç”
AdSoyad = Ad + “ “ + Soyad

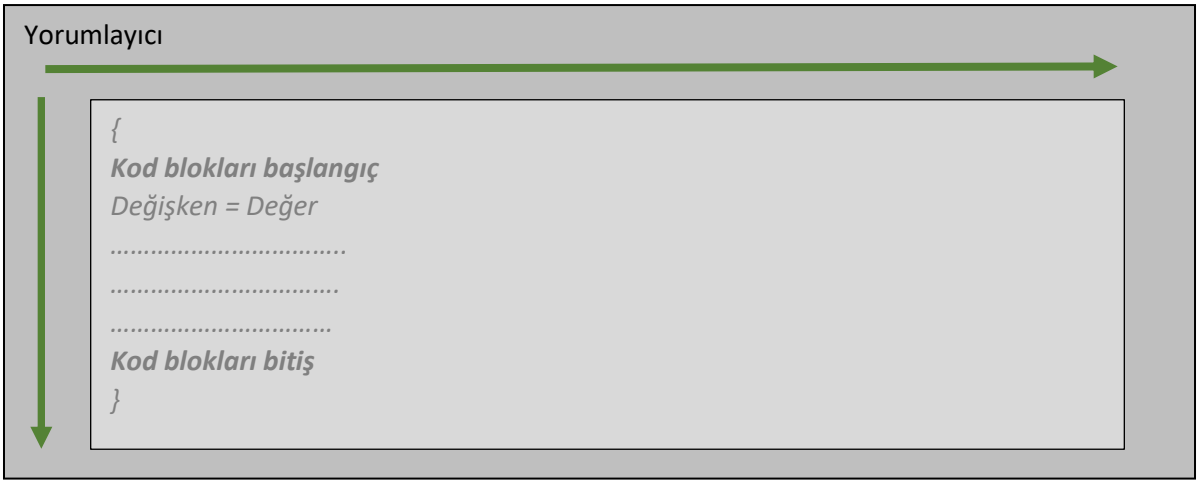
Bu örnek sonucunda AdSoyad isimli değişkenin son değeri “tunç güleç” olacaktır.

Değişkenlerin Ömürleri: Bu konuya geçmeden önce yorumlayıcılar hakkında kısa bir bilgi vermek istiyorum. Yorumlayıcılar (Compiler) yazdığımız komutları bilgisayarın anlayacağı dile çeviren, oradan da elektronik seviyeye kadar indiren derleyicilerdir. Bizim öğreneceğimiz C# dilinin yorumlayıcısı .net framework isimli yorumlayıcıdır. (.)Net FrameWork yorumlayıcısı yazdığımız komutları yukarıdan aşağı ve soldan sağa doğru okur. Bu bilgi değişkenlerin ömürlerini anlamamızda yardımcı olacaktır.



Değişkenlerin ömürleri kod bloğunun başladığı yerden, bittiği yere kadardır. 4. Gizli Mesaj: Değişkenlerin ömründen bahseder misin? Ancak uzun süreli ve geliştirilen yazılımın açık olduğu süre boyunca hayatta kalan değişkenler de vardır.

Aşağıdaki yorumlayıcı desenine bakacak olursak değişkenin ömrünü ifade eden yorumlayıcı deseni şu şekilde olacaktır. Değişken "***Kod blokları başlangıç"*** olan yerde başlar, "***Kod blokları bitiş"*** olan yerde biter. Yani bellekten atılır.



Bu konumuzda size, bir önceki derste anlattığım değişkenlerin devamı olan değişken tiplerini anlatacağım.

Bir önceki dersimizden hatırlayacağınız gibi, değişkenlere değer ataması gerçekleştirmeden önce, o değişkenin hangi tipte bir değeri tutacağını belirmek gereklidir (istisnai durumlar hariç).

Yine önceki derslerimden hatırlayacağınız üzere, değişken tanımlarken dikkat etmeniz gereken bazı kurallar söz konusu. Bu kurallar programınızın sağlıklı sonuç elde etmesi anlamında hayati öneme sahiptir.

1. Tanımlanan değişkenin alabileceği minimum ve maksimum değeri bilmek.
   Bu kural şu anlama gelmektedir: Özellikle sayısal değer alan değişken tiplerinde minimum ve maksimum değerler vardır. Bu değerler yazılın dilinin standartlarına göre gelmektedir ve değiştirilemezler.
   ***Örnek:** Alabileceği değerler -9 ile 9 arasında olan bir değişken tipine -10 verdiğinizde programlama diline göre program ya hata verecektir ya da aşağı yukarı yuvarlanacaktır. C# dili bu tür durumlarda hata verip programınızın derlenmesini engelleyecektir. Ancak bu işlem çalışma zamanında meydana gelirse program hata verip kapanacaktır. Değişken tipleri minimum maksimum değerleri ek olarak ders materyali olarak eklenmiştir.*

2. Tanımlanan değişkenin ömrünü bilmek.
   Bu kural şu anlama gelmektedir: Bir önceki dersimizde anlatıldığı üzere değişkenin belleğe yerleştiği ve bellekten atıldığı anı bilmek.

3. Tanımlanan nerelerden erişileceğini bilmek.
   Bu kural şu anlama gelmektedir: Tanımlanan değişkenin hangi kod bloklarından erişebileceğini bilmek, o değişkene atanan değere ulaşmak için önemlidir. Bu durum çoğu zaman birden fazla kez değişken oluşturma yükünden kurtaracaktır.

4. Değişken tanımlama metodolojisini bilmek.
   Bu kural şu anlama gelmektedir: Tanımlanacak değişkenin yazılım dili kurallarına uygun olarak belirtilmesi anlamına gelir. Yazılım dilinin kuralına göre yazılmayan her komut seti gibi, yanlış komutlar ile tanımlanan değişkenler de hata verip programın derlenmesini engelleyecektir.

5. Değişkene ihtiyacını bilmek.
   Bu kural şu anlama gelmektedir: Her ne kadar değişken tanımlamakta özgür davranılsa da, her bir değişkenin, tanımlandığı tipin minimum ve maksimum değerler arasında bellekte bir yer kapladığı bilinmeli ve gereksiz değişken tanımlanmamalıdır.

Bu konumuzda size değişken tanımlama metodolojisini, kurallarını anlatacağım.

Önceki derslerde, her yazılım dilinin yazım kuralları olduğundan ve bu kurallar çerçevesinde yazılan kodların nasıl derlendiğinden bahsetmiştik.

Yazdığımız komutların derleyiciler tarafından bilgisayarların anlayacağı dile çevrildiğini biliyoruz. Derleyicilerin bu komutları anlaması için, her yazılım dilinin kod yazma standartları vardır. Bu standartlardan Syntax Kuralları isimli derste bahsetmiştim.

Kurallar: Değişken tanımlamak için dilin tarafımıza dayattığı yaptırımlardır. Bunlar başlıca şöyledir.

- Değişken isimlerinde latin alfabesi kullanılmalıdır. Her ne kadar c# buna izin verse de, "ĞÜŞÇİı" karakterler kullanılmamalı.
  Örnek: Ağaç, Şeker, Üst, Şan, İsim, ışık

- Değişken ismi özel karakter ve noktalama işareti içeremez.
  Örnek: Ad~Soyad, Ad-Soyad, Ad\Soyad, Ad/Soyad

- Değişken isimleri rakam ile başlayamaz.
  Örnek: 1_Ogrenci, 2_Veli, 1İnsan

- Değişken isimleri, birden fazla kelimeden oluşuyorsa aralarında boşluk olamaz.
  Örnek: Ad Soyad, Veli Ad

- Değişken isimleri içerisinde harf, rakam ve alt çizgi dışında karakter kullanılamaz.
  Örnek: Ad:Soyad, Veli-Ad, Ogrenci*Ad

- Değişken tanımlarken kolay okunabilir olması ve notasyonlara uygun olarak yazılması gerekir.

Bahsettiğim bu kurallara istinaden doğru değişken isimlendirmesi ve tanımı şöyle olacaktır.

Örnek: AdSoyad, adSoyad, Ad_Soyad

Değişkenler tanımlanırken aynı tipte tanımlanan değişkenler için kolay yöntemler vardır.

Örnek olarak şu tanımlama yöntemini verebiliriz.

int a,b,c,d,f,g;
Görüldüğü üzere aynı tipe sahip olan değişken isimlerini, aralarına virgül koyacak şekilde tanımlayabiliriz.

Peki varsayılan değerlere sahip aynı tipte değişken tanımlamak zorunda kalsaydık. O zaman kodumuz şu şekilde olacaktı. 5. Gizli Mesaj: Dizi tanımlama kurallarından bahseder misin?

int a = 3, b = 55, c = 99, d =66;

Başka bir değişken tanımlama yöntemi de **var** tanımı ile yapılmaktadır. **var** terimi ile tanımlanan değişkenler genellikle hangi tipte değer tutacağı bilinmeyen zamanlarda kullanılmaktadır. Her tipte değeri tutabileceği gibi alışkanlık hale getirilmemelidir. Çünkü standart bir değişken tanımlama yöntemi değildir. 6. Gizli Mesaj: Yukarıdaki değişkenlerden b' nin değeri kaçtır.
var terimi ile tanımlanan değişken mutlaka bir başlangıç yani varsayılan değer almak zorundadır. Çünkü bellekte tutacağı yere kendisine atanan değerin tipine bakarak karar vermektedir. Değişkene verilen değer kendisi için referans olmaktadır.

Metodoloji:
var DegiskenAdi = Varsayılan Değer;

Bu konumuzda size erişim belirleyicileri anlatacağım.

Erişim belirleyici nedir: Erişim belirleyiciler (Access modifiers) adından da anlaşılacağı üzere, tanımlanan bir objenin, erişime izin verilmesi istenen özelliğini temsil eder. Erişim belirleyicisi tanımlanan objeye erişim izni, kısıtlaması getirmek için kullanılan bir ayardır.

Object Oriented programlama dillerinde her objenin bir nesne olarak değerlendirildiğinden ve kendisi de nesneden türeyen bir objeye ait olması gerektiğinden bahsetmiştik. Kavram kargaşasını azaltmak adına şu örneği vermek istiyorum.

```
TemelNesne
{
        AltNesne
        {
        }
        Obje
        {
        }
}
```

Örnekte görüldüğü üzere her nesne veya obje temel bir nesnenin parçası olmak zorundadır. Erişim belirleyiciler tam olarak burada karşımıza çıkmaktadır. Alt nesnelerin veya temel nesnelerin, diğer nesneler veya objeler tarafından erişime açık olup olmadığının belirlenmesi veya bellekte ne zamana kadar duracaklarının ataması erişim belirleyiciler ile yapılır.

Erişim Belirleyiciler:

- **Public:** Public olarak tanımlanan bir objeye kod bloğunun içinden ve dışından erişime açıktır. Her yerden erişilebilir.
- **Protected:** Protected olarak tanımlanan bir objeye, bu oluşturulan objeden türetilmiş başka obje tarafından erişilebilir.
- **Internal:** Internal olarak tanımlanan bir objeye sadece bağlı bulunduğu projeden erişilebilir.
- **Private:** Private olarak tanımlanan bir objeye bulunduğu obje içerisinden erişilebilir.
- **Protected Internal:** Protected internal olarak oluşturulan bir objeye, tanımlandığı objeden, türetilmiş başka objeden ve dış uygulamalardan da erişilebilir.

Erişim Belirleyici Metodolojisi:

[ErişimBelirleyicisi] [Tip] [ObjeAdi] = [İsteğe Bağlı Varsayılan Değer];

C# Örneklendirmeleri:

Public:
public string OgrenciAdi = “Tunç Güleç”;
public int OgrenciNo;
public bool Gecti = true;
public bool Kaldi;

Protected
public string VeliAd = “Tunç Güleç”;
public int VeliGSM;

7. Gizli Mesaj: Yukarıdaki değişkenlerden Gecti isimli değişkenin başlangıç değeri kaçtır?

```
TemelNesne
{
        Public string AdSoyad;
        Public string VeliAdSoyad = “Tunç Güleç”;

        Obje
        {
        Public string OgrenciAdSoyad;
        Public int OgrenciNo= 1111;
        }

        metod
        {
        Public string DogumYeri =”Istanbul”;
        Public int DogumYili= 1979;
        }

}
```

Erişim belirleyiciler sadece bir objenin parçası iken kullanılabilirler. Bir metot içinde kullanılamazlar. Erişim belirleyiciler ile ilgili doğru ve yanlış tanımı anlatan örnekte inceleyebilirsiniz.

Bu konumuzda size tipler arasında dönüşümleri anlatacağım.

Tip dönüşümleri; herhangi bir tipten diğer tiplere dönüştürmek için kullanılan yöntemlerdir.

Kullanımı oldukça basittir. Neredeyse tüm IDE araçlarında olan IntelliSense (Kod Tamamlama) özelliği ile kolayca yapılmaktadır. IDE'nin IntelliSense özelliğini başlatmak için CTRL+Space tuşuna basarak ardından "Convert" yazarak tip dönüşümlerini gözlemleyebilirsiniz.

Convert.To[DönüştürülecekTip](DönüştürülecekDeğer);

Başlıca Tip Dönüşümleri:

- ToBoolean: Parametre ile alınan değeri Boolean mantıksal tipine çevirir. Çeviremediğinde hata verir.
- ToByte: Parametre ile alınan değeri Byte tipine çevirir. Çeviremediğinde hata verir.
- ToChar: Parametre ile alınan değeri Char tipine çevirir. Çeviremediğinde hata verir.
- ToDateTime: Parametre ile alınan değeri DateTime tipine çevirir. Çeviremediğinde hata verir.
- ToDecimal: Parametre ile alınan değeri Decimal tipine çevirir. Çeviremediğinde hata verir.
- ToDouble: Parametre ile alınan değeri Double tipine çevirir. Çeviremediğinde hata verir.
- ToInt32: Parametre ile alınan değeri Int32 tipine çevirir. Çeviremediğinde hata verir.
- ToInt16: Parametre ile alınan değeri Int16 tipine çevirir. Çeviremediğinde hata verir.
- ToString: Parametre ile alınan değeri string tipine çevirir. Çeviremediğinde hata verir.

Bu tip dönüşümlerinin yanı sıra Type Casting ismiyle anılan bir yöntem daha vardır. "Miş" gibi davranmak adıyla da yorumlanabilir. 8. Gizli Mesaj: Type casting yönetimi bana anlatır mısın? Normal dönüşüm komutları, dönüşüm hatalarında runtime anında hata verirken, bu yöntem kodlama zamanında da hata verir. Bu yöntem nesneler ve daha çok sayısal değişkenler için kullanılmaktadır.

Kullanımı:
([Dönüşüm yapılacak tip])Değer;

Bu konumuzda size dizileri yani array anlatacağım.

İşe, dizilerin ne olduklarını anlatarak başlayalım.

Aynı tipte birden fazla değişken tanımlamak yerine kullanılan değişkenler bütünüdür. Buna şöyle diyebiliriz. Aynı tipte "n" tane aynı değere veya farklı değerlere sahip değişkenlerin bir arada tutulduğu tümleşik değişken grubudur.

Şu örneği inceleyelim.
int a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,……..,z;

İsminde değişkenlerimiz olsun. Birde bunlara teker teker değer atayalım.
a=1;
b=3;
c=99;
d=42;
…….
z=76;

Değişken tanımlama ve değer atamanın ne hayli zor olduğunu görebiliyoruz. İşte diziler bu gibi durumlarda ve daha fazlasında imdadımıza yetişiyor.

Diziler; runtime anında yeni eleman kabul edebiliyor ve eksiltebiliyorlar. Değerlerine erişim sağlanabiliyor ve güncellemeler yapabiliyoruz.

Dizi Tanımlama Metodolojisi:

ErişimBelirleyicisi Tip[] DiziAdi = new Tip[Eleman Sayısı];

Bunun kod tarafında yazım hali şu şekilde olmalı.

```
public int[] Sayilar = new int[100];
```

Bu örnekte görüleceği üzere Sayilar isminde, int tipinde 100 elemanlık bir dizi oluşturulmuş oldu. Yani 1 tane, 100 adet değişkene sahip değişkenler grubuna sahibiz.

Bu durumu şöyle yorumlayabiliriz.

İnt Değişkenim

- Değişkenimin 0. Alt değişken
- Değişkenimin 1. Alt değişken
- Değişkenimin 2. Alt değişken
- Değişkenimin 3. Alt değişken
- Değişkenimin 99. Alt değişken

Açıklamaya dikkatinizi çekmek istiyorum. Dizilerde, alt değişkene yani değişkenin elemanına index numaraları ile ulaşabiliyoruz. Bu da şu anlama geliyor; dizi elemanlarına yani alt değişkenlere ekleniş sırasına göre erişebiliyoruz. Buradan da anlaşılacağı üzere ekleniş sırasına index deniyor.
Ayrıca indexlerin sıra numaraları 0'dan başlıyor. Evet diziler sayı saymaya 0'dan başlar. Dolayısıyla bir dizinin ilk elemanı 0, son elemanı da eleman sayısı -1 olmaktadır. 9. Gizli Mesaj: 99 Elemanlık bir diziye ait son elemanın index numarası kaçtır?
Bizim örneğimizde ilk eleman sıra numarası 0, son elemanı ise 99 dur.

Dizinin elemanlarına ulaşmak için [] (köşeli parantez) işaretleri arasında eleman index numarası, yani

İnt Değişkenim

- Değişkenim[0]. Alt değişken
- Değişkenim[1].Alt değişken
- Değişkenim[2].Alt değişken
- Değişkenim[3]. Alt değişken
- …….
- Değişkenim[99]. Alt değişken

sıra numarasını kullanmalıyız. Şu tabloyu inceleyerek durumu özetleyebiliriz.

Dizinin elemanlarına değer atamak yine değişkenlere değer atamada olduğu gibi, atama operatörü olan eşittir (=) işareti ile yapılmaktadır.

Dizim[SıraNumarası] = Değer;

Kod bloğumuz şunun gibi olmalı.

```
int[] sayilar = new int[100];
sayilar[0] = 312;
……;
sayilar[99] =5421;
```

Dizinin elemanlarından değer almak için ise dizinin eleman sıra numarasını bir değişkene atamak yeterli olacaktır.

[Tip] DegiskenAdi;
DegiskenAdi = Dizim[SıraNumarası];

Kod bloğumuz şunun gibi olmalı.

```
int[] sayilar = new int[100];
int SayiVer;
sayilar[0] = 312;
……;
sayilar[99] =5421;
SayiVer = sayilar[0];
```

Kod bloğunda göründüğü üzere "**SayiVer**" isimli değişkenin son değeri, "**sayilar**" isimli dizinin 0.

Elemanının değeri olan 312 değeri olmaktadır. 10. Gizli Mesaj: Bahsedilen elemanın değeri kaç olmalıdır?

Diziler ile ilgili framework'un bir çok hazır metodu ve özelliği mevcuttur. Bunlarda bazılarına kısaca değinmek istiyorum.

Yine intellisense (kod tamamlama) özelliği sayesinde bu metotlara ve özelliklere ulaşabilirsiniz. Bu metotlara ulaşmak için "Array" özel kelimesi ile intellisense tamamlamasını bekleyebilirsiniz.

[DiziAdı].Length : Dizinin o anki eleman sayısını veren özelliktir.
Array.Clear : Dizini elemanlarının verilerini temizler.
Array.Copy: Kaynak diziyi, hedef diziye, eleman sayısı kadar kopyalar.
Array.Find: Kaynak dizi içerisinde aranan dizi elemanını bulur.
Array.FindAll: Kaynak dizi içerisinde aranan dizi elemanlarını bulur ve başka bir diziye atama yapar.
Array. FindIndex: Kaynak dizi içerisinde aranan dizi elemanının index numarasını yani sıra numarasını verir.
Array.FindLast: Kaynak dizi içerisinde arama yaparak, bulduğu benzer dizi elemanlarından sonuncusunu getirir.
Array.FindLastIndex: Kaynak dizi içerisinde arama yaparak, bulduğu benzer dizi elemanlarından sonuncusunun sıra numarasını yani index numarasını getirir.
Array.FindIndex: Kaynak dizi içerisinde arama yaparak, bulduğu elemanın sıra numarasını yani index numarasını getirir.
Array.IndexOf: Kaynak olarak verilen dizi elemanları içerisinde, verilen değeri arar ve bulduğu elemanın index numarasını yani sıra numarasını döndürür. Eğer birden fazla eleman bulursa, bulduğu ilk elemanın sıra numarasını döndürür. Eğer bir eleman bulunamazsa -1 değeri döndürür.
Array.LastIndexOf: Kaynak olarak verilen dizi elemanları içerisinde bulunan elemanlardan sonuncusunun index numarasını verir. Eğer bir eleman bulunamazsa -1 değeri döndürür.
Array.Resize: Kaynak olarak verilen diziyi, verilen yeni sayı kadar yeniden boyutlandırır. Eğer verilen sayı, dizi sayısından küçükse son elemanlardan başlayarak siler.
Array.Reverse: Kaynak olarak verilen diziyi tersine çevirir.
Array.Sort: Kaynak olarak verilen diziyi sıralar.

Dizilerin en çok merak edilen konularından biri de çok boyutlu dizilerdir. Çok boyutlu dizileri anlamak için kısaca bir tabir vardır. Dizi içinde dizi.

Kullanımı yine dizilere benzer. Aradaki tek fark x dizisinin içindeki elemanın da artık bir dizi olmasıdır. Şu örnekte iç içe dizi, dinamik dizi kullanımını gözlemleyebilirsiniz.

```
tip[] dizim = new tip[2];
  * dizim[0] = new tip[3];
      dizim[0][0] = değer;
      dizim[0][1] = değer;
      dizim[0][2] = değer;
  * dizim[1] = new tip[3];
      dizim[1][0] = değer;
      dizim[1][1] = değer;
      dizim[1][2] = değer;
```

```
int[] dizim = new int[2];
  dizim[0] = new int[3];
    dizim[0][0] = 1001;
    dizim[0][1] = 1002;
    dizim[0][2] = 1003;
  dizim[1] = new int[3];
    dizim[1][0] =2001;
    dizim[1][1] =2002;
    dizim[1][2] =2003;
```

Bu kullanımın diğer kullanımı için çoklu boyutlu dizi ifadesini kullanmak yanlış olmaz.

Çok boyutlu diziler:
Tanımlanma metodolojisi şu şekildedir.
tip[,] diziAdi = new tip[İlkBoyutunElemanSayısı,İkinciBoyutunElemanSayısı];
Köşeli parantezler içine dikkat çekmek istiyorum. Tip tanımı yapılan köşeli parantezler içindeki virgül sayısı+ 1, dizinin kaç boyutlu olacağını sembolize eder. Diziyi oluşturduğumuz new komutunun yanındaki köşeli parantezler içindeki virgül ise her dizi boyutunun kaç adet elemana sahip olacağını sembolize eder. Virgülün solundaki ilk boyutun eleman sayısı, sağındaki diğer dizinin boyutunu temsil eder.

Kod bloğu şu şekilde olmalı.

```
tip[] dizim = new tip[2];
  * dizim[0] = new tip[3];
      dizim[0][0] = değer;
      dizim[0][1] = değer;
      dizim[0][2] = değer;
  * dizim[1] = new tip[3];
      dizim[1][0] = değer;
      dizim[1][1] = değer;
      dizim[1][2] = değer;
```

```
int[] dizim = new int[2];
  dizim[0] = new int[3];
    dizim[0][0] = 1001;
    dizim[0][1] = 1002;
    dizim[0][2] = 1003;
  dizim[1] = new int[3];
    dizim[1][0] =2001;
    dizim[1][1] =2002;
    dizim[1][2] =2003;
```

int[,] sayilar = new int[2,2];

Bu konumuzda size listlerden bahsedeceğim.

Listler adında da anlaşılacağı üzere, liste demektir. Listeler tüm tiplere destek vermektedir. Bir liste başka bir listeyi barındırabilir. İleride göreceğimiz derslerde Lambda denen arama yöntemleri ile içinde arama yapılabilir. Listler "generic" denen bir namespace' e bağımlılık duyar ve projenize varsayılan olarak gelmezler. Bu nedenle List kullanmak ihtiyacımız olduğunda, kod satırlarının en üstünde yer alan "Using" alanında "`System.Collections.Generic`" `namespace'in eklenmesi gerekmektedir.`

List metodolojisi:
[List<tip>] [ListeAdı] = new List<tip>([Opsiyonel Liste Eleman Sayısı]);

Kod bloğumuz şunun gibi olmalı.
```
List<int> sayilar = new List<int>(100);
```

Listeler oldukça akıllı nesnelerdir. Hem az yer kaplarlar hem de kullanılmayan liste elemanları yer kaplamazlar. Akıllı sıralama ve indexleme yöntemleri vardır.
Listelere eleman eklemek, çıkarmak ve liste değerlerine ulaşmak kolay ve hızlıdır.

.Net framework tarafından liste işlemlerini hızlıca yapabileceğiniz hazır metodlar ve özellikler vardır. Bu metodlara ve özelliklere ulaşmak için liste adını yazıp .(nokta) tuşuna basarak intellisense'in tamamlamasını bekleyebilirsiniz.

ListeAdi.Add : Listeye yeni eleman eklemek için kullanılır. Listeye eklenecek eleman, oluşturulan tipte bir değer olmalıdır.
ListeAdi.AddRange : Listeye yeni elemanlar (toplu olarak) eklemek için kullanılır. Listeye eklenecek elemanların, oluşturulan liste tipte bir değer olmalıdır.
ListeAdi.Capacity: Listenin kapasitesini vermektedir.
ListeAdi.Clear: Listeyi boşaltmak için kullanılmaktadır.
ListeAdi.Contains : Listenin içinde parametre olarak verilen başka bir elemanı aramak için kullanılır. Liste içinde arama yapılması istenen eleman, oluşturulan liste tipinde olmalıdır.
ListeAdi.CopyTo : Listeyi başka bir nesneye kopyalamak için kullanılmaktadır.
ListeAdi.Count : Listenin eleman sayısını vermek için kullanılmaktadır.
ListeAdi.Exists : Listenin içinde parametre olarak verilen nesnenin varlığını kontrol eder. Bulursa true, bulamazsa false değerini döndürür.
ListeAdi.Find: Listenin içinde parametre olarak verilen nesneyi arar ve bulduğu ilk elemanı döndürür.
ListeAdi.FindAll : Listenin içinde parametre olarak verilen nesneyi arar ve bulduğu elemanları liste halinde geri döndürür.
ListeAdi.FindIndex : Listenin içinde parametre olarak verilen nesneyi arar ve bulduğu ilk elemanın index numarasını geri döndürür. Eğer listenin içinde aranan eleman yoksa -1 döner.
ListeAdi.FindLast : Listenin içinde parametre olarak verilen nesneyi arar ve bulduğu son elemanı döndürür.
ListeAdi.FindLastIndex : Listenin içinde parametre olarak verilen nesneyi arar ve bulduğu son elemanın index numarasını geri döndürür. Eğer listenin içinde aranan eleman yoksa -1 döner.
ListeAdi.Remove : Liste içinden parametre olarak verilen elemanı silmek için kullanılır. Eğer aynı elemandan birden fazla bulunması durumunda bulunan ilk eleman silinir. Silme başarılı ise true, değilse false döndürür.

ListeAdi.RemoveAll : Liste içinden parametre olarak verilen elemanı silmek için kullanılır. Eğer aynı elemandan birden fazla bulunması durumunda bulunan tüm elemanlar silinir. Silme başarılı ise true, değilse false döndürür.
ListeAdi.RemoveAt : Liste içinden, parametre olarak verilen index numarasına ait elemanı silmek için kullanılır.
ListeAdi.RemoveRange : Liste içinden, parametre olarak verilen ilk değerden başlayarak, yine parametre olarak verilen ikinci değer kadar eleman siler.
ListeAdi.Reverse: Listeyi tersine çevirmek için kullanılır.
ListeAdi.Sort: Listeyi sıralamak için kullanılır.
ListeAdi.ToArray: Listeyi diziye çevirmek için kullanılır.
ListeAdi.Insert: Listeye index numarası verilerek eleman eklemek için kullanılır. Listeye eklenecek eleman, oluşturulan tipte bir değer olmalıdır.
ListeAdi.Last : Listenin içinde parametre olarak verilen nesneyi arar ve bulduğu son elemanın kendisini verir.
ListeAdi.LastIndexOf : Listenin içinde parametre olarak verilen nesneyi arar ve bulduğu son elemanın index numarasını geri döndürür. Eğer listenin içinde aranan eleman yoksa -1 döner.
ListeAdi.Remove : Listenin içinden parametre olarak verilen elemanı siler. Parametre olarak verilen nesne, liste tipinde olmalıdır. Elemanın silinirse true silinemezse false değeri döner.
ListeAdi.RemoveAll : Liste içinde arama yaparak bulduğu elemanları silmek içi kullanılır.
ListeAdi.RemoveRange : Liste içinden, parametre olarak verilen index değerinde itibaren, yine parametre olarak verilen adet kadar silmek için kullanılır.
ListeAdi.Reverse : Listeyi ters çevirmek için kullanılır.

Bu konumuzda size enumlardan bahsedeceğim.

Enum; değişkenlerin alabileceği değerleri sınırlandırmak için kullanılan bir nesne tanımlama yöntemidir. 11. Gizli Mesaj: Listeyi diziye çevirmek için kullanılan yöntem nedir? Öyle bir değişken düşünün ki sadece programcı tarafından belirlenen değerleri alabilsin.

Örneğin: string bir değişken tanımladığımızı düşünelim. Bu değişkenin sadece belli değerlerden oluşabileceğini anlatmanın bir yolu yok.

Şu örneğe bakalım.

string Cinsiyet;
Cinsiyet = “Erkek”;
Cinsiyet = “Kadın”;
Cinsiyet = “Herhangi”;

Bu örnekte olduğu gibi Cinsiyet isimli değişkene string ifade olacak şekilde istediğimiz değeri atayabiliyoruz. Oysa öyle bir yöntem olmalı ki, değişkenin değerine önceden belirlenmiş değerler dışında hiçbir değer verilemesin. İşte enum tam olarak bu işe yarıyor.

Enum tanımlama metodolojisi: 12. Gizli Mesaj: Liste içinden, parametre olarak verilen index numarasına ait elemanı silmek için kullanılan yöntem nedir?

[Erişim Belirleyici] enum [Enum Adı] { Kadın = [Opsiyonel İndex Numarası], Erkek = [Opsiyonel Index Numarası] };

Kod bloğumuz şunun gibi olmalı.
public enum Cinsiyet {Kadin =1 , Erkek = 2};

Opsiyonel olarak index numarası verilmemesi halinde soldan sağa 0 dan başlayarak index numaraları atanır.

Şimdi oluşturulan enum tipinde bir değişken oluşturalım ve neler olduğuna bakalım. 13. Gizli Mesaj: Bana enumlardan bahseder misin?
Cinsiyet cins;
cins = Kadin;

Bu kod satırında “Cins” isimli değişkene herhangi bir string değer atamaya çalıştığınızda hata almanız kaçınılmazdır.

Enum tiplerinde oluşturulan bir değişkene index numaraları yani int veriler kontrol edilebilmektedir. Ancak bu int veri enum değerlerine verilen numaralarından oluşmalıdır.

Enumlar, karşılaştırma ve karar yapılarında oldukça etkilidirler. Herhangi bir değerin string ifade ile karşılaştırması oldukça dikkat ister. 14. Gizli Mesaj: Listenin eleman sayısını vermek için kullanılan yöntem nedir? Çünkü küçük büyük harf duyarlılığı olan yazılım dillerinde hata payı vardır. Çünkü ifade edilen “tunc” ile “Tunc” aynı değildir. Bu iki karşılaştırma sonucunda çalışan bir kod bloğu, aradaki farktan dolayı çalışmayacaktır. Enumlar karşılaştırma operatörlerinde, karar yapılarında hata yapmazlar. Çünkü belirtilen veriler dışında değer girmek mümkün değildir.