Problem Çözme Süreci

Özet

Bu Bölümde;
Problemlerin keşifsel ve algoritmik çözümleri arasındaki farkları belirtebilecek,
Algoritmik bir çözümü olan problemleri çözmek için gereken 6 problem çözme adımını açıklayabilecek,
Bir problemi çözebilmek için 6 problem çözme adımını kullanacak,
Değişken ve Sabit arasındaki farkı algılayacak,
Karakter, Sayısal ve Mantıksal veri türlerini anlayacak,
Operatör, işlemci ve sonucu ayırt edecek,
Fonksiyon tanımlayıp kullanacak,
İlişkisel ve mantıksal operatörleri kullanacak,
İşlem önceliğini kavrayacak,
İfade ve eşitlikleri kullanarak işlem yapacaksınız.

Ali GÖKAŞAR
Bilişim Teknolojileri Öğretmeni

İçindekiler

I. Problem Çözme Teknikleri

Problemleri çözerken çözüm sürecine destek olan bazı yaklaşımları kullanırız. Programlama sürecinde de problemin çözümüne yönelik yol ve yaklaşımları belirlemek gerekir ama öncelikle genel kural ve teknikleri bilmek yararlıdır. Bazı genel kurallar neredeyse tüm problemler için kullanılabilir. Bu nedenle, bu kuralları içselleştirir ve düşünme sürecinizin bir parçası haline getirebilirsek herhangi bir problemi çözmeye çalışırken mutlaka bir fikrimiz olacaktır.

i. Her Zaman Bir Planınız Olsun

Belirsiz bir durumu yaşamak yerine her zaman bir planınız olmalıdır. Bu, en önemli kuraldır.

Belki oluşturduğunuz çözüm planı ilk denemelerde sonuç vermeyecek ama her seferinde sizi çözüme bir adım daha yaklaştıracak ipuçları elde etmenizi sağlayacaktır. Denediğiniz yaklaşım ne olursa olsun (doğru ya da yanlış), fikir üretmemekten ve deneme yapmamaktan her zaman daha iyidir.

Planlama yapmak aynı zamanda hedef belirlemek ve bu hedefe ulaşmak anlamına gelir. Planınız olmazsa tek bir hedefiniz olur; o da problemi çözmektir. Bu durumda problemi çözene kadar herhangi bir başarı kaydetmiş olmazsınız.

Dahası sadece nihai hedefe ulaşmaya çalışmak, her başarısızlıkta moralin bozulmasına neden olabilir. Ama küçük hedeflerden bile oluşan bir plan yaparsanız çözüme yönelik adımlar attığınızı ve zamanı etkili biçimde kullandığınızı göreceksiniz.

Her bir adımda kendinize güveniniz artacak ve çözüme bir adım daha yaklaşmış olacaksınız.

ii. Problemi Tekrar İfade Edin

Bazen problemi tekrar ifade etmek, **göremediğimiz bir ayrıntıyı görmemizi ya da problemi daha kolay çözmek adına bir ipucu yakalamamızı** sağlayabilir. Hatta bazen probleme ilişkin bir yanlış anlamanın ortaya çıkmasına ya da hedefin daha iyi anlaşılmasına neden olur.

Problemi farklı biçimlerde sunmak çözüm sürecine ışık tutmasa bile bazen yalnızca problemi doğru anlayıp anlamadığımızı teyit etme açısından önemlidir. Ayrıca tekrar ifade ederek problemi küçük alt parçalara ayırmak gibi yaygın işlemleri de kolaylaştırmış oluruz.

iii. Problemi Küçük Parçalara Ayırın

Verilen problemi adımlara ya da bölümlere ayırmak, çözümü kolaylaştıracaktır.

Bir problemi iki bölüme ayırdığımız düşünüldüğünde, her bir parçanın çözümünün tümünü çözmeye göre yarı yarıya kolaylaştığını düşünebiliriz.

Bu durum için sıralama örneğini ele alalım. Elinizde 100 kişisel dosya olduğunu ve bu dosyaların alfabetik sıraya göre dizilmesi gerektiğini varsayalım.

Önce bir dosya alıp sonra her aldığını doğru yere yerleştirerek 100 adımda bu işlemi bitirebilirsiniz.

Peki ya biri sizin için 100 dosyayı A-F, G-M, N-S ve T-Z olacak biçimde 4 gruba ayırsaydı işlemi daha kolay yapmaz mıydık?

Her seferinde tek tek harf sırasını kontrol etme işlemi, dosya sayısı çoğaldıkça zorlaştığından sayıca küçük bir grupla çalışmak işlemi oldukça hızlandıracaktır. Bu nedenle bir problemi çözülebilir küçük parçalara bölmek, çözüm işlemini kolaylaştırır ve hızlandırır.

iv. Önce Bildiklerinizden Yola Çıkın

Programlama yaparken öncelikle bildiklerimiz ile başlamalı ve sonra yeni çözümler arayışına girmeliyiz.

Problemi küçük parçalara bölerek çözebildiğiniz parçadan başlamalısınız. Bu parçaları çözerken diğer parçalarla ilgili olarak aklınıza yeni fikirler geldiğini ve aynı zamanda kendinize olan güvenin arttığını göreceksiniz.

Programlama süreci boyunca çoğu zaman çok iyi olduğunuz konular, zorlandığınız konular ve henüz öğrenmediğiniz konular olacaktır. Bir problemin mevcut becerilerle çözülüp çözülemeyeceğine karar vermek için problemi çok iyi incelemeniz gerekmektedir. Böylece bilgi dağarcığınızda olmayan ama çözüm için gerekli olan işlemlerin farkına varabilirsiniz.

v. Problemi Basitleştirin

Çözmekte zorlandığınız bir problemle karşılaşırsanız problemin **kapsamını daraltmayı deneyin**. Bunun için koşulları azaltmayı ya da çözebileceğiniz biçime dönüştürmeyi, değişkenleri azaltmayı ya da problemin kapsama alanını küçültmeyi düşünebilirsiniz.

Temel amacınız problemi basitçe ifade etmeye çalışmak olmalıdır.

Çözüm için denediğiniz yaklaşımlar, size gerçek çözüm için yol gösterecektir. Problemi basitleştirmek size aslında problemdeki zorluğun neden kaynaklandığını da gösterecektir.

vi. Benzerlik Arayın

Burada ele aldığımız benzerlik kavramı, **çözülmesi istenen problemle önceden çözülen problem arasındaki olası örtüşme ya da yeni çözüme ilham verme** olarak tanımlanabilir.

Benzerlik, farklı biçimlerde karşımıza çıkabilir. Bazen problemler aynı, değişkenler ya da veriler farklıdır. Bazen de problemin bir bölümü başka bir problemle benzerlik gösterebilir.

Problem çözme sürecinde hızınızı ve becerinizi arttıracak en önemli yaklaşım; benzerliklerin farkına varmaktır. Ancak bu, aynı zamanda kazanılması en zor beceridir. Buradaki zorluk, benzerliklerin farkına varacak kadar problem çözmek ve deneyin kazanmak gereğinden ortaya çıkmaktadır.

Bu nedenle programlamaya yeni başlayanlar için bir problemi çözerken hazır bir kodu bulmak ve onu güncelleyerek problemi çözmeye çalışmak son derece yanlıştır.

Bir çözümü kendiniz üretmezseniz tamamen anlayamaz ve içselleştiremezsiniz. Ayrıca çalışan her bir program, problemin çözümü olduğu kadar, sonraki problemleri çözmek için kazandığınız deneyim olacaktır.

Yeni başlarken hazır yazılmış kodlar kullanarak problemi çözerseniz gelecekte de sürekli bu yaklaşımı kullanacaksınız demektir.

vii. Deneme Yapın

Bazen bir problemi çözmenin en kolay yolu denemek ve sonuçlarını gözlemlemektir.

Bu tahmin etmekten çok farklıdır. Bir çözümü tahminen öngörmek ile deneyip gözlemlemek farklı sonuçlar verir. Böylece problemi çözebilmek için gereken ipuçlarını elde edebilirsiniz.

Denemek, ara yüz tasarlarken, çizim yaparken ya da kütüphaneleri kullanırken yararlı bir yaklaşımdır. Diğer yandan hata ayıklama süreci de bir tür deneysel yaklaşımdır. Bir problemin çıktılarına bakarak sorunun nereden kaynaklandığını anlayabilir ve problemi çözebiliriz. Programda önemli noktalara gözlemek amacıyla değer verebiliriz. Değerlerin değişimini gözleyerek çözüme daha hızlı ulaşabiliriz.

viii. Asla Vazgeçmeyin

Asla vazgeçmemek **kişisel bir özelliktir.**

Kararlılık, güven ve istek kaybolduğu zaman açık düşünemezsiniz, işlemler olması gerektiğinden uzun sürer ve gittikçe zorlaşır. Hatta öfke ve kızgınlığa bile dönüşebilir.

Ekrandaki program kodu çalışmadığı zaman yazılımcı koda değil, kendisine ve aslında problemin kaynağı olan kendi aklına kızmaktadır. Bu noktada moralimizin bozulmasına izin vererek başarısız olmak için bahane üretmiş oluruz. Bu duygudan kurtulmak, yazılımcı tarafından verilecek bir karardır.

Böyle durumlar en etkili çözüm ara vermektir. Problemden tamamen uzaklaşarak geçirilecek vakit sonrasında daha verimli çalıştığınızı göreceksiniz.

II. Problem Çözme Adımları

Problem çözme sürecinde en iyi kararı verebilmek için izlenmesi gereken 6 adım vardır:

Problemi Tanımlama

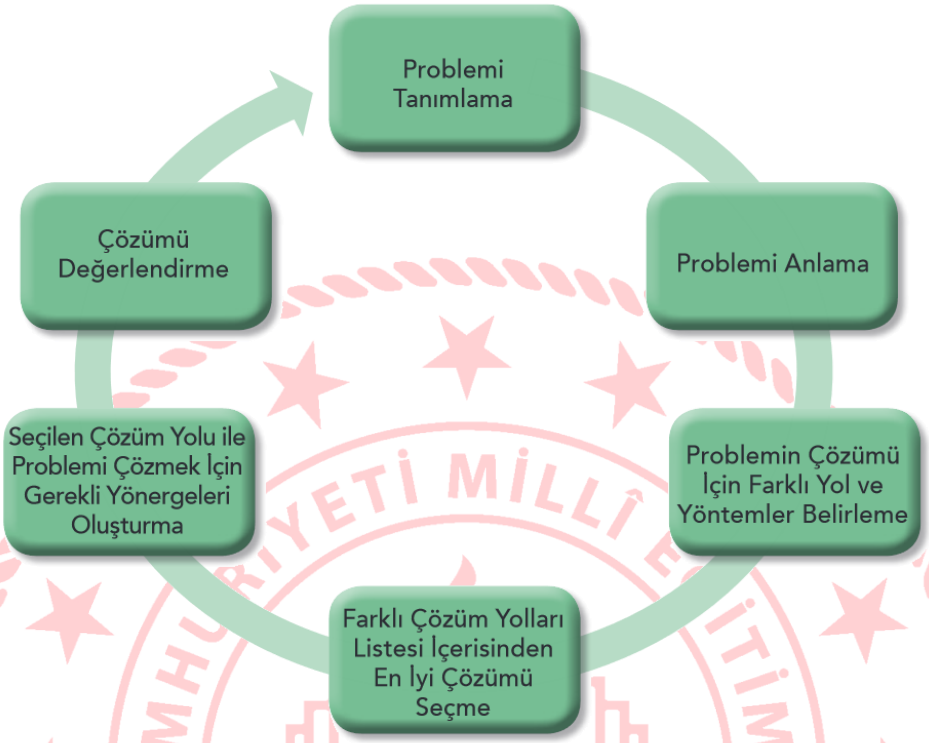
Problemi Anlama

Problemin Çözümü için Farklı Yol ve Yöntemler Belirleme

Farklı Çözüm Yolları Listesinden En İyi Çözümü Seçme

Seçilen Çözüm Yolu ile Problemi Çözmek için Gerekli Yönergeleri Oluşturma

Çözümü Değerlendirme



i. **Problemi Tanımlama:** Problemi çözmeye başlamadan önce problemin açık, anlaşılır ve çok doğru bir şekilde tanımlanmış olması gerekir.

Problemin ne olduğunu bilemezseniz onu çözemezsiniz.

ii. **Problemi Anlama:** Çözüme doğru yol almadan önce problemi çok iyi anladığınızdan emin olmanız gerekir. Problemin neler içerdiğini ve kapsamını doğru anlamalısınız. Ayrıca problemi çözmeniz gereken insan ya da sistemin bilgi tabanında neler olduğunu da çok iyi anlamalısınız. Mevcut bilgi tabanında olmayan herhangi bir kavram ya da yönergeyi problemin çözüm sürecinde kullanamazsınız.

Bu konuda klasik ve önemli bir söz vardır:
"Problemi anlamak, problemi yarı yarıya çözmek demektir."

iii. **Problemin Çözümü için Farklı Yol ve Yöntemler Belirleme:** Problemin çözümü için olabildiğince farklı yol ve yöntem belirlemeli ve bu listenin tüm olasılıkları içerdiğinden emin olmalısınız. Bunun için konu hakkında farklı kişilerin görüşlerini alabilirsiniz. Farklı çözümler kabul edilebilir olmalıdır.

Problem çözmek için tek bir yol yoktur; pek çok yol vardır.

iv. **Farklı Çözüm Yolları Listesi İçerisinden En İyi Çözümü Seçme:** Bu adımda her bir çözümün olumlu ve olumsuz yönlerini ortaya koymalısınız. Bu nedenle değerlendirme yapabilmek için ölçütler oluşturmalısınız. Bu ölçütler her bir çözüm yolunu değerlendirmek için size rehber olacaktır.

Problem çözmek için tek bir yol yoktur; en iyi yol vardır.

v. Seçilen Çözüm Yolu ile Problemi Çözmek için Gerekli Yönergeleri Oluşturma: Bu adımda numaralandırılmış ve adım adım yönergeler oluşturmanız gerekir. Bu yönergelerin ikinci adımda belirtilen bilgi tabanı kapsamında olmasına dikkat ediniz. Bu durum, özellikle bilgisayarlar ile çalışırken son derece kısıtlı davranmanıza neden olabilir.

vi. Çözümü Değerlendirme: Çözümü test etmek ya da değerlendirmek, sonucun doğruluğunu kontrol etmek anlamına gelir. Sonucun doğru olması ve problemi olan bireyin beklentilerini karşılama düzeyi önemlidir. Sonuç yanlış çıkmış ya da bireyin beklentilerini karşılamamış ise problem çözme sürecine baştan başlamak gerekir.

Problem çözme sürecinde bu 6 adım tam olarak uygulanmaz ise sonuç beklendiği gibi olmayabilir.

Bireyler olarak evde, işte ya da farklı ortamlarda sürekli problem çözeriz. Evde karşılaşılan problemler; akşam yemeği için ne pişirileceği, yemekten sonraki boş vaktin nasıl değerlendirileceği, hangi kitabın okunacağı ve marketten nelerin alınacağı gibi konulardır. İş yerinde ise problemler; yönetim, iş yeri kuralları ve takımların verimli çalışması gibi konuları içerebilir.

Ne kadar doğru kararlar alırsanız o kadar mutlu ve değerli yaşantılar geçirirsiniz. Çoğu kişi bu süreçlerde problem çözdüğünün bile farkına varmaz.

III. Problem Türleri

Problemlerin her zaman sıradan çözümleri olmaz. Kek yapmak ya da araba kullanmak gibi problemleri çözmek için bir dizi eylem gerekir. Adım adım yönergelere dayalı olan bu çözümlere "algoritmik çözümler" denir. En iyi yolu seçtikten sonra sonuca, ilgili adımları izleyerek ulaşılır. Bu adımlardan oluşan yapıya "algoritma" denir.

En lezzetli ekmeği seçmek ya da işleri büyütmek için yatırım yapmak gibi problemlerin ise açık ve net ifade edilen yanıtları yoktur. Bu çözümler bilgi ve deneyim gerektirir, bir dizi deneme ve yanılma sürecinden oluşur. Doğrudan işlem adımları ile ulaşılamayan sonuçlara "keşfe dayalı çözümler" denir.

Problemi çözen kişi her iki türdeki problem için problem çözmenin 6 adımını kullanabilir. Ancak keşifsel çözümler için son adım çok doğru sonuç vermeyebilir. Bazı problemler ise her iki türdeki çözümün de kullanılmasını gerektirir.

IV. Bilgisayarlar ile Problem Çözme

Bu ders kapsamında **çözüm** demek problem çözme sürecinin 5. adımında yer alan işlem adımları ya da yönergeler anlamına gelmektedir.

Sonuç demek, çıktı ya da tamamlanmış bilgisayar destekli yanıt demektir.

Program ise herhangi bir bilgisayar dilinde kodlanmış, çözümü oluşturan işlem adımlarının tamamını ifade etmektedir.

Bilgisayarlar, zor ve zaman alıcı olabilen algoritmik çözümler ile ilgilenmek üzere tasarlanmıştır. İnsanlar, keşifsel çözümleri bulma konusunda daha iyidirler ancak bilgisayarların çözebildiği ileri düzey hesaplama ve karmaşık problemleri çözme konusunda bilgisayarların hızlarına ulaşamazlar. Bilgisayarlar, üst düzey matematik problemlerini kolayca çözebilir ancak konuşmak ya da top atmak gibi davranışları yapamaz. Bu işlemlerin zorluğu programlama sürecindedir. Bu tür işlemleri bilgisayarların anlayacağı bir dizi adım şekline nasıl dönüştürebiliriz?

Keşifsel problem türleri ile ilgilenen bilgisayar dalına **yapay zekâ** adı verilmektedir. Yapay zekâ uygulamaları, bilgisayarlara mevcut bilgileri kullanarak yeni bilgiler inşa etmesini sağlamaktadır. Böylece bilgisayarın problem çözme becerileri insanların yeteneklerine daha çok benzemektedir. Yapay zekâ özellikle robotik uygulamaları ile son yıllarda popüler bir çalışma alanı olmuştur. Bilgisayarlar insanlar gibi düşünmeye başlayana kadar daha çok algoritmik problemlerin çözüm süreçlerinde kullanılacaktır. Bu nedenle bu derste algoritmik çözümler üzerinde durulacaktır.

V. Problem Çözme Kavramları

Günlük hayatta karşılaştığımız problemler çok çeşitli olmasına rağmen bilgisayar ile çözebildiğimiz yalnızca 3 tür vardır:

Hesaplamalı-Matematiksel işlem ve süreçler içeren problemler

Mantıksal-İlişkisel süreçler içeren problemler

Tekrarlayan-matematiksel ya da mantıksal bir dizi işlemin yinelenme sürecini içeren problemler

Bu bölümde, bilgisayara ilişkin temel kavramlar ve belirtilen türdeki problemleri çözmek için kullanılan ifade ve eşitlikler anlatılacaktır. "Sabit" ve "değişken" olmak üzere iki önemli kavram vardır.

Programcı işlenmemiş hâlde veriyi alır, işlenmiş hâle yani bilgiye dönüştürür. Bunlar eşitlik ve ifadelerin yapı taşlarıdır. Programcı, problemi çözebilmek için gereken sabit ve değişkenleri, uygun veri türünde, örneğin sayısal olarak tanımlar.

Diğer önemli kavramlar ise operatör ve fonksiyonlardır. "Operatör", sabit ve değişkenler arasındaki ilişkileri gösteren, eşitlik ve ifadelerde kullanılan işaret ve sembolleri ifade eder. Operatörlerin belirli bir hiyerarşik yapı içerisinde kullanılması gerekir. Operatörler sabit ve değişkenlerle birlikte kullanıldığında "eşitlik" ve "ifade" olarak adlandırılan yapılar oluşur. Eşitlik ve ifadeler ise çözüm sürecinin yapı taşları olan işlemlerdir.

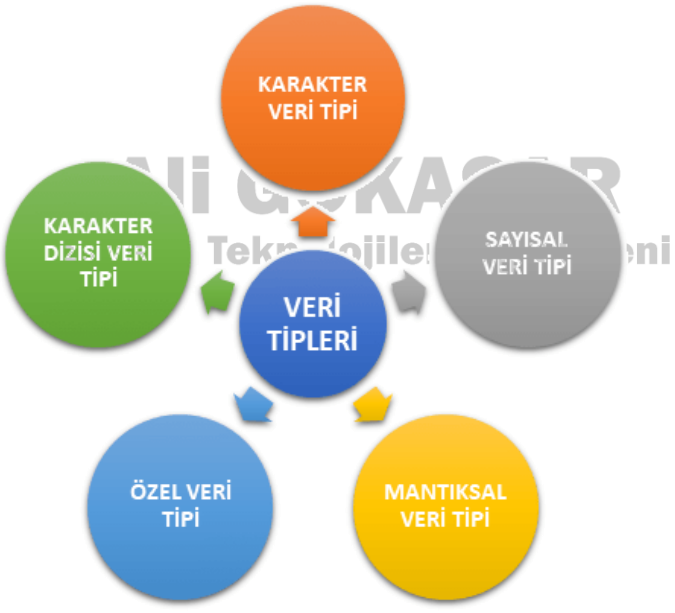
"Fonksiyonlar" bir dizi işlem seti olarak tanımlanabilir. Tüm bu kavramları anlamadan bilgisayarlar ile problem çözmek olası değildir.

VI. Veri Türleri

Çevremizdeki kavram ve nesneleri farklı şekillerde anlamlandırmak için farklı veri türleri kullanırız. Çözümler üretebilmek için bilgisayarlar veriye gereksinim duyar.

Ham veriler, bilgisayar tarafından "girdi" olarak algılanır ve program aracılığı ile işlenir. Kullanıcıya geri dönen değer, işlenmiş veridir; "çıktı" ya da "bilgi" olarak adlandırılır.

Bilgisayara hangi veri türüyle çalışıyor olduğunu mutlaka belirtmeniz gerekmektedir. Bir programda farklı veri türleriyle işlem yapılabilir. Örneğin tam sayılar, kesirli sayılar, karakterler, simgeler, metinler ve mantıksal değerler, veri türlerini oluşturur.



i. Sayısal Veri

Sayısal veriler tüm sayı tiplerini içerir. **Sayısal veri, hesaplama işlemlerinde kullanılabilen tek veri türüdür.** Pozitif ya da negatif tam sayılar ve reel sayılar kullanılabilir.

Sayısal veriler; açılar, uzaklık, nüfus, ücret, yarıçap gibi hesaplama sürecinde gerekli değerler için tanımlanır. Banka hesap numarası ya da posta kodu gibi sayısal ama hesaplama için kullanılmayan veriler de vardır. Bu tür veriler sayısal olarak tanımlanmaz.

Her bir veri türünün bir veri seti vardır. Sayısal veri için tanımlanmış veri seti 0-9 arasındaki sayılar ve "+" ile "–" işaretlerini kapsar. Örneğin 66578 ve -2356 tam sayı örnekleridir. Reel sayılar, tüm reel ve ondalık sayıları kapsar. Örneğin, 56.23, 8695.235 ya da 0.005 reel sayı için örneklerdir. Sayıların alabileceği en küçük ve en büyük değerler kullanılan bilgisayar ve programlama diline göre değişebilir.

Veri Türü	Veri Seti	Örnek
Sayısal: Tam Sayı	Tüm Sayılar	55326
Sayısal: Reel Sayı	Tüm Reel Sayılar ve Ondalık Sayılar	56.23 0.005

ii. Alfanümerik/Karakter Veri

Karakter veri seti; tüm tek haneli sayılar [0-9], harfler (a-z, A-Z) ve özel karakterleri (#, &, *, ..) kapsar. **Bu veri setinden oluşturulan değer, tırnak içinde belirtilir.**

Büyük ve küçük harf duyarlıdır yani "a" ile "A" farklı algılanır. ASCII (American Standard Codefor Information Interchange) olarak adlandırılan karakter seti 256 karakterden oluşur. Karakterler sadece sayıdan oluşsa bile **hesaplama işlemlerinde kullanılamaz.**

Birden fazla karakter bir araya getirilirse bilgisayar, bu yapıyı "dizi" olarak adlandırır. Karakter ve dizi verileri karşılaştırılabilir ve alfabetik sıraya göre sıralanabilir. Bilgisayar her karaktere bir numara verir ve işlemi bu şekilde gerçekleştirir çünkü bilgisayarlar sayısal işlem yapabilen cihazlardır. Veriler birbirleri ile karşılaştırılır ve azalan ya da artan şekilde sıralanır.

Örneğin Muz ile Elma karşılaştırıldığında M harfi E harfinden daha büyük bir sayıya sahip olduğu için Muz dizisinin değeri daha büyüktür. Elif ile Esra karşılaştırıldığında ise Esra daha büyük değer alır çünkü s harfi l harfinden daha sonra gelir. Büyük harflerin küçük harflerden daha düşük sayısal değerleri vardır.

Veri Türü	Veri Seti	Örnek
Karakter	Tüm Rakamlar, Harfler ve Özel Semboller	"G", "F", "L", "2", "8", "+"
Dizi	Birden Fazla Karakterden Oluşan Kombinasyon	"Bilgisayar" "123-456789"

Karakterler ve diziler + operatörü kullanılarak birbirine bağlanabilir. Birleştirme olarak adlandırılan bu işlem, iki karakter parçasını yan yana getirir. Örneğin "6"+"6" = "66" olur. Genellikle matematiksel işlem gerektirmeyen verilerin, dizi şeklinde tanımlanması önerilir.

iii. Mantıksal Veri

Mantıksal veri, veri setinde yalnızca iki kelime barındırır: **Doğru** ve **Yanlış**. Bu veri Evet ya da Hayır şeklindeki karar verme süreçlerinde kullanılır.

Örneğin elde edilen değer, beklenen değer mi, evli mi, arabası var mı, öğrenci lise mezunu mu gibi sonucu kesin doğru ya da yanlış olan durumlarda mantıksal veri tanımlaması yapılır. Bu kelimeler ayrılmış özel kelimelerdir ve dizi olarak algılanmaz.

Veri Türleri İçin Kurallar

Tanımladığınız veri genellikle sayısal, karakter, dizi ya da mantıksal olmalıdır.

Programcı programlama sürecinde verinin adını ve türünü belirtir. Bilgisayar çalışmaya başladığında verinin adı ile türünü eşleştirir.

Veri türleri karışık kullanılamaz. Örneğin sayısal olarak tanımlanmış bir veri, dizi olarak algılanamaz. Bu durumda bilgisayar, beklediği veri türü ile karşılamaz ve hata verir.

Her bir veri türü kendisi için tanımlı veri setini kullanır

Matematiksel işlemlerde kullanılacak tüm veriler sayısal olarak, diğerleri karakter ya da dizi olarak tanımlanmalıdır.

Programcılar kendi tanımladıkları veri türlerini de oluşturabilirler. Kullanıcı tanımlı olarak adlandırılan bu veri türleri, bugünün tarihi, hedef, varılacak süre gibi hem dizi hem de sayısal veriler içeren yapılar oluşturulabilir.

Veri	Veri Türü	Açıklama
Ürün Satış Bedeli: **49.99**	**Sayısal: Reel**	Bir ürünün satış bedeli hesaplama işlemlerinde kullanılır.
T.C. Kimlik No: **10000000146**	**Karakter Dizisi**	Kimlik No hesaplama amaçlı kullanılmaz.
Ağırlık: **82**	**Sayısal: Tamsayı**	Kilo cinsinden tamsayı olabilir ve hesaplamalarda kullanılabilir.
Kredi Onayı: **Var, Yok**	**Mantıksal**	Bu durumda onay ya vardır (Doğru) ya da yoktur (Yanlış).
IBAN: **TR2800001122000033**	**Karakter Dizisi**	Para transferi için bankaya verilen kodlar hesaplama amaçlı kullanılmaz.

VII. Bilgisayar Veriyi Nasıl Saklar?

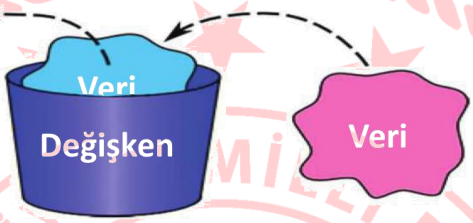
Bilgisayar veriyi hafızada saklar. Her bir **değişken** için hafızada belirli bir alan ayrılır ve bu alan her seferinde tek bir değer saklayabilir.

Kullanıcı, var olan değer yerine yeni bir değer atadığında eski değer silinir. Hafızada bu konumlar **geçicidir**. Programın çalışması bittiğinde ya da bilgisayar kapatıldığında bu veriler silinir. Verilerin daha sonra tekrar kullanılması gerekiyorsa sabit disk gibi kalıcı bir konuma kaydedilmeleri gerekir. Bu şekilde kaydedilen verilere **dosya** adı verilir.

Temel anlamda program dosyaları ve veri dosyaları olmak üzere iki dosya türü vardır. Program dosyaları, bilgisayarın yapması istenen komutları ve işlemleri içerir. Veri dosyaları ise programlar çalışırken gereken verileri kapsar.

VIII. Sabit ve Değişkenler

Bilgisayarlar problemleri çözmek için süreç boyunca **sabit** ve **değişken** olarak adlandırılan verileri kullanır. **Sabit** olarak tanımlanan veriler problemin çözüm süreci boyunca asla değişmeyen değerlerdir. Sabit değerler sayısal, karakter ya da özel semboller olabilir. Bu durumda bu değere bilgisayarın hafızasında bir yer ayrılır ve bir isim verilir. Program çalıştığı sürece bu değer kendisine verilen isim ile çağrılır ve değeri asla değiştirilemez. Örneğin, pi değeri değişmeyen bir değer olacağı için sabit olarak tanımlanmalıdır.



Bu durumun tam tersi şekilde bir "değişken" tanımlandığında değeri, program çalıştığı sürece değişebilir. Değişkenlere taşıdığı değerleri ifade eden isimler verilir, bu şekilde belirleyici özellikleri de oluşur.

Programcılar çözüm sürecinde ihtiyaç duyulan her bir değişkene ayrı bir isim vermelidir. Böylece bilgisayar bu ismi, ilgili değeri hafızada bulmak için kullanır. Değişken, farklı veri türlerinde olabilir ancak ismi, içerdiği değer ile tutarlı olmalıdır. Örneğin fiyat isimli bir değişenin içerisinde 50 değeri atanmış olabilir, program çalıştığı süre içerisinde bu değer değişebilir ancak değişkenin ismi hiçbir zaman değişmez.

Değişkenlere isim verirken ve bunları kullanırken dikkat edilmesi gereken kurallar şunlardır:

Değişkene içerdiği değer ile tutarlı isimler veriniz

Değişkenlere isim verirken boşluk kullanmayınız.

Değişkenlere isim verirken bir karakter ile başlayınız.

Matematiksel semboller kullanmamaya dikkat ediniz

Değişken isimleri konusunda aşağıdaki noktalara dikkat edilmelidir:

Bazı platformlar desteklemediği için Türkçe karakter kullanımı tavsiye edilmez.

Programlama dillerinde kullanılan komut isimleri değişken olarak kullanılamaz. Örneğin; if, for, while, else, do, int, vb.

Özel karakterler değişken isimlerinde kullanılamaz (*, /, +, #, %, &, (, =, $, [, { vb.)

Değişken isimlendirmelerinde boşluk karakteri yerine alt çizgi (_) karakteri kullanılabilir ancak değişken isimlendirmede genellikle küçük harfle başlanır ve ikinci bir kelime yazılacaksa ilk kelimenin hemen ardından büyük harfle devam edilir. Buna "Camel Karakter" kullanımı denir. Örnek: tcKimlikNo

IX. Fonksiyonlar

Fonksiyonlar, **belirli işlemleri yürüten ve sonuçları döndüren bir işlem kümesidir.** Genellikle bilgisayar dilinde oluşturulur. Fonksiyonlar, bir çözüm sürecinin belirli parçaları olarak kullanılır. Problem çözme sürecinde tekrarlanan işlemler için kullanılır ve böylece programcının hem problemi daha hızlı çözmesini hem de programın daha anlaşılır olmasını sağlar.

Her programlama dili, içerisinde kendine özgü fonksiyonlar barındırır. Bu fonksiyonlar kütüphanesi, programlama diline göre değişiklik gösterir. Ayrıca pek çok programlama dili, programcıların kendi fonksiyonlarını yazmalarına da olanak verir. Fonksiyon kütüphaneleri, pek çok program diline eklenebilir.

Fonksiyonlar, kendilerine verilen isim ve ayraç içerisinde gönderilen veri ile tanımlanır.

FonksiyonIsim(Veri)

Fonksiyon kapsamında elde edilen sonuç, fonksiyonun ismi ile döndürülür. Fonksiyonlara veri gönderilir. Fonksiyona gönderilen verilere **parametre** denir. Fonksiyonlar parametreleri değiştirmez ama işlemlerde kullanır.

Örneğin karekök fonksiyonunu ele alalım. Sqrt(N), gönderilen N değeri için karekök değeri hesaplamaktadır. Sqrt fonksiyonun ismi, N işlem yapılacak veri yani parametredir. Parametreler yay ayraç içinde yazılır. Programcı olarak kullandığınız dilin kütüphanesinde hangi fonksiyonların olduğunu araştırmanız işlerinizi kolaylaştıracaktır. Fonksiyonlar gruplara ayrılır:

Matematiksel Fonksiyonlar: Matematiksel işlemler için kullanılır.

Dizi Fonksiyonlar: Dizi ve karakterle ilgili işlemleri gerçekleştirmek için kullanılır.

Dönüştürme Fonksiyonları: Veriyi bir türden diğerine dönüştürmek için kullanılır.

İstatistiksel Fonksiyonlar: Maksimum değer, ortalama gibi değerleri hesaplamak için kullanılır.

Yardımcı Fonksiyonlar: Program dışındaki verilere erişerek işlem yapmak için kullanılır.

X. Operatörler

Bilgisayara, verileri nasıl işleyeceğini belirtmek gerekir. Bu işlem için **operatörler** kullanılır. **Operatörler** verileri, ifade ve eşitlikler ile birleştirir. Bu yazım, aynı zamanda operatörler bilgisayara ne tür bir işlem (matematiksel, mantıksal vb.) olduğuna dair bilgi verir.

İşlemci ve sonuç, operatörlere ilişkin iki kavramdır. İşlemci, verileri bağlayan ve işleme alan yapı; sonuç ise yapılan işlemin yanıtıdır. Örnek vermek gerekirse 6 + 5 ifadesinde yer alan + **operatör**, 6 ve 5 **işlemci**, 11 ise **sonuçtur**. İşlemciler sabit ya da değişken olabilir. Operatörler; matematiksel, mantıksal ve ilişkisel operatörler olarak sınıflandırılabilir. Operatör türlerine ilişkin örnekler aşağıdaki tabloda yer almaktadır.

Operatör	Bilgisayar Sembolü	Örnek	
Matematiksel		İşlem	Sonuç
Toplama	+	6+2	8
Çıkarma	-	5-3	2
Çarpma	*	3*4	12
Bölme	/	8/4	2
Mod Alma	%	9%4	1
İlişkisel		İşlem	Sonuç
Eşittir	==	1==1	True
Küçüktür	<	2<3	True
Büyüktür	>	2>3	False
Küçük veya Eşittir	<=	2<=3	True

Büyük veya Eşittir	**>=**	2 <= 3	False
Eşit Değildir	**<>**	2 <> 3	True
Mantıksal		İşlem	Sonuç
Değil	**NOT**	NOT True	False
Ve	**AND**	True AND False	False
Veya	**OR**	True OR False	True

İlişkisel operatörlerle yapılan işlemlerin sonucunda ortaya mantıksal değer olarak Doğru (True) ya da Yanlış (False) çıkar.

XI. İfade ve Eşitlikler

Şu ana kadar gördüğümüz tüm bileşenler, ifade ya da eşitlik biçiminde kullanılmadığı sürece bir anlam ifade etmez. Çözülmeye çalışılan problem vergi ya da maaş hesaplama, değerleri sıralama, en büyük değeri bulma gibi farklı işlemlerden oluşabilir. Bir ifade operatörleri kullanarak veriyi işler.

Uzunluk * Genişlik (Bir ifadedir fakat sonucu saklanmaz.)

"Eşit" operatörü kullanılarak ifadenin sonucu başka bir değişkende saklanır.

Alan = Uzunluk * Genişlik

Bu durumda uzunluk ve genişlik değerlerinin çarpım sonucu hafızada **alan** olarak ayrılan yerde korunur. İfadelerde eşit operatörü kullanılmaz. İfadeler eşitlik ve yönergelerin yalnızca bir bölümünü oluşturur. Bu yüzden sonuçlar o an kullanılır ancak korunmaz. Oysaki eşitlik ifadelerinde mutlaka sonuç korunur. Bu yüzden eşitliklere **atama ifadeleri** de denir.

İfadeler	Eşitlikler
A+B A ve B sayısal veridir. Sonuç sayısaldır ve hafızada saklanmaz.	**C = A+B** A, B ve C sayısal veridir. Sonuç sayısaldır ve C değişkenine atanarak korunur.
A < B A ve B sayısal, karakter ya da dizi olabilir. Sonuç mantıksal değerdir ve hafızada saklanmaz.	**C = A < B** A, B ve C sayısal, karakter ya da dizi olabilir. Sonuç mantıksal değerdir ve C değişkenine atanarak korunur.