

TEMEL KAVRAMLAR

Bilgisayar: Kullanıcıdan aldığı veriler üzerinde hızlı bir biçimde aritmetiksel ve mantıksal işlemler yapabilen, yaptığı işlemlerin sonucunu saklayabilen; sakladığı verilere istenildiğinde tekrar ulaşılabilmesine imkan sağlayan elektronik bir alettir.

Bilişim Teknolojileri: Bilginin üretilip insanların hizmetine sunulana kadar geçen süreçte kullanılan her türlü teknolojiye denir.

Teknoloji: İnsanoğlunun tasarlayarak ürettiği ya da uygulamaya koyduğu faydalı veya faydasız her türlü alet ve araçlardır.

Veri: Doğruluğu kanıtlanmamış bilgidir.

Bilgi: Öğrenme, araştırma ya da gözlem yoluyla elde edilen, insan zekasının çalışması sonucu ortaya çıkan düşünce ürünüdür.



DONANIM ve YAZILIM

Bir bilgisayar sistemi iki ana unsurdan meydana gelir.

Donanım: Bilgisayarların fiziksel parçalarına denir. *Örnek: Fare, Klavye, Monitör*

Yazılım: Bilgisayarı belirli işlevleri yerine getirmek üzere yöneten, bilgisayara ne yapacağını söyleyen, kodlanmış komutlar dizisidir. **Program** da denir.

YAZILIM ÇEŞİTLERİ

İşletim Sistemi Yazılımları: Bilgisayar donanımının doğrudan denetimi ve yönetiminden, temel sistem işlemlerinden ve uygulama programlarını çalıştırmaktan sorumlu olan ana yazılımdır. *Örnek: Windows7, Windows10, MacOS, Linux*

Uygulama Yazılımları: Belirli konulardaki problemlerin çözümüne yönelik olarak programlama dillerinden biri ile yazılmış programlardır. Hangi işletim sistemine uygun olarak yazılmışsa o işletim sistemi altında çalışırlar. Değişik amaçlara yönelik binlerce uygulama yazılımı vardır.

Örnek:Office programları (Word,Excel,Powerpoint vs.), photoshop, winrar, oyunlar, antivirüs yazılımları, telefonlarımızdaki uygulamalar vs.

! İster genel amaçlı isterse özel amaçlı olsun tüm uygulama ve sistem yazılımları programlama dilleriyle yazılır. Bir programlama dili, insanların bilgisayara çeşitli işlemler yaptırmasına olanak sağlayan her türlü simge, karakter ve kurallar grubudur. Programlama dilleri insanlarla bilgisayar arasında çevirmenlik görevi yapar. Programlama dilleri, bilgisayara neyi, ne zaman, nasıl yapacağını belirten deyim ve komutlar içerir.

PROGRAMLAMA KAVRAMLARI

Programlama: Bilgisayar programlarının yazılması, test edilmesi ve bakımının yapılması sürecine verilen isimdir.

Programlama Dili: Programcının bilgisayara hangi veri üzerinde işlem yapacağını, verinin nasıl depolanıp iletileceğini, hangi koşullarda hangi işlemlerin yapılacağını tam olarak anlatmasını sağlar. Bir programlama dili, insanların bilgisayara çeşitli işlemler yaptırmaya olanak sağlayan her türlü simge, karakter ve kurallar grubudur.
Örnek: Python, C#, C++, Java, Swift, GoLang

Yazılımcı: Bir programlama dili kullanarak, yazılım (program) üreten kişilerdir.

Algoritma: Belli bir problemi çözmek veya belirli bir amaca ulaşmak için tasarlanan ve izlenecek olan yoldur.

Neden Programlama Öğrenmeliyiz?

Bilgisayara bir işi yaptırırken yani programlarken düşünmeyi öğreniriz. Analiz yapabilme ve problem çözebilme yeteneğimiz artar. Bir sorunla karşılaştığımızda daha iyi düşünebiliriz ve o sorun için bulduğumuz çözümlerin sayısı artar. Olaylara daha ayrıntılı bakabiliriz. Bilgisayarda program yazarken algoritmalar geliştiririz.

Algoritma problemin adım adım çözülme sürecidir. Bu da bize olayları detaylı düşünebilme ve çözebilme yeteneği kazandırır.

DONANIM

Bilgisayar donanımları üç ana birimden oluşur.

Sistem Birimi: Ana kart, merkezî işlem birimi, ana bellek, ekran kartı, sabit disk, disket sürücü, kasa ve güç ünitesi gibi donanımların bulunduğu birimdir.

Giriş Birimleri: Bilgisayara veri aktarılmasını sağlayan birimlerdir.

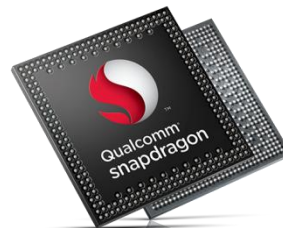
Örnek: Fare, klavye, tarayıcı, mikrofon vs.

Çıkış Birimleri: İşlemcinin çıkan sonuçları aktarabileceği birimlerdir.

Örnek: Monitör, yazıcı, hoparlör vs.

TEMEL DONANIM BİRİMLERİ

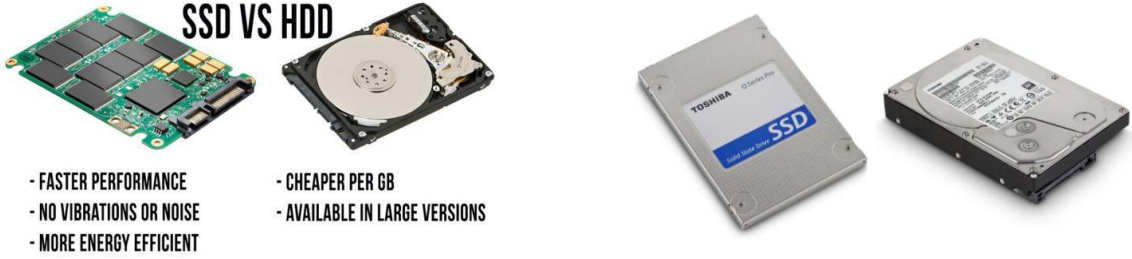
Merkezi İşlemci Birimi (CPU): Bilgisayarda aritmetik ve mantık işlemlerinin yapıldığı ve bunların denetlendiği merkezdir. Yani işlemci, bilgisayarın beynidir.



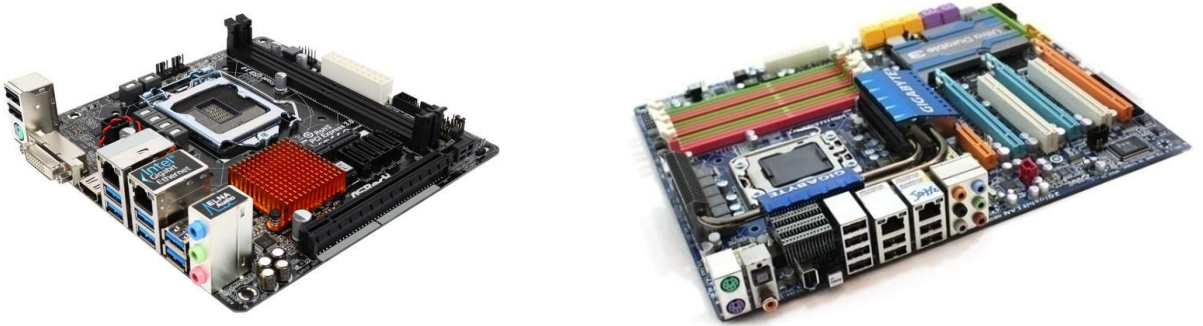
RAM Bellek (Random Access Memory): Bilgisayarda çalışmakta olan bir programa ait komutlar ve veriler ile daha sonra kullanılacak olan sonuç işlemlerinin geçici olarak saklandığı donanımdır.



Sabit Disk (Hard Disk): Bilgisayarda tüm bilgilerin depolandığı donanımdır. Günümüzde artık mekanik hard disklerin (HDD) yerine SSD (Solid State Disk) diskler kullanılmaya başlanmıştır. HDD'lere göre daha sessiz çalışması, daha az enerji tüketmesi ve daha hızlı çalışmasından ötürü SSD'ler günümüzde hızla yaygınlaşmaktadır.



Anakart (Main Board): Bilgisayarın tüm iç ve dış donanım birimlerinin üzerindeki bağlantı portlarına bağlandığı, bu donanımlar arasındaki iletişimi sağlayan, üzerinde elektronik devre elemanlarının bulunduğu bilgisayarın en temel parçasıdır.



Ekran Kartı: Bilgisayarın görüntü vermesini sağlayan donanımdır. Belirli bir bellek ve işlemci kapasitesi kullanarak grafiksel hesaplamayı yapan donanımdır. Harici olabileceği gibi anakart üzerinde tümleşik olanları da mevcuttur.



Güç Kaynağı (Power Supply): Bilgisayarın çalışması için elektrik enerjisini sağlayan birimdir. Genellikle metal bir kasanın içerisinde bulunur. Güç kaynağı elektriği prizden alarak onu bilgisayarın kullanabileceği 5V ve 12V değerine ayarlar.



Klavye (Keyboard): Klavye, üzerinde harf, rakam, özel karakterler ve özel fonksiyon tuşlarının bulunduğu bir bilgisayar giriş birimidir. Bilgisayar temelde klavye aracılığıyla yönlendirilir ve kumanda edilir.

Fare (Mouse): Ekrandaki öğeleri seçmenizi ve hareket ettirmenizi, bu öğelerin temsil ettiği işlemleri (sol tuş ile bir veya iki kez tıklayarak) yaptırmanızı sağlayan bir giriş aygıtı'dır.

Yazıcı (Printer): Bilgisayardaki verilerin kağıt üzerine basılmasını sağlayan bir çıktı aygıtıdır.

Tarayıcı (Scanner): Elimizdeki bir dokümanın görüntüsünü bilgisayar ortamına aktarmamızı sağlayan bir girdi aygıtıdır.

Hoparlör (Speaker): Elektrik dalgalarını ses dalgalarına çeviren ve gerektiğinde sesi yükseltebilen çıktı aygıtıdır.

Mikrofon: Ses dalgalarını elektriksel titreşimlere çeviren donanımdır. Bu sayede sesimizi bilgisayar ortamına aktarabiliriz.

Monitör: Monitör (veya ekran) bilgisayarın mikroişlemcisinden gönderilen sinyalleri gözün görebileceği şekilde görüntüye dönüştüren cihazdır. Yani CPU tarafından işlenilen bilgilerin kullanıcıya iletildiği ortamdır; bir çıkış birimidir.

Taşınabilir Depolama Araçları: Bilgisayar dışındaki ortamlarda verilerimizi saklamamıza olanak sağlayan donanımlardır. Bunlar; harici disk, flash bellek, hafıza kartı, CD, DVD, Blue Ray Disk, vb. dir.



! Bu donanımların kullanılabilir hale gelmesi için işletim sistemine tanıtılması gerekmektedir. Donanımları işletim sistemine tanıtan yazılımlara **driver** (sürücü) denir.

ETİK DEĞERLER

Etik: Bireylerin ahlaklı ve erdemli bir hayat yaşayabilmesi için hangi davranışlarının doğru, hangilerinin yanlış olduğunu araştıran bir felsefe dalıdır.

Bilişim Etiği: Bireylerin bilişim teknolojilerini ve interneti kullanımı sırasında uymaları gereken kuralları tanımlayan ilkelere bilişim etiği denir.

Fikri Mülkiyet: Kişinin kendi zihni tarafından ürettiği her türlü ürün olarak tanımlanmaktadır.

Fikri ve kültürel eserlerden bazıları **Creative Commons (CC)** organizasyonuna dahildir. Creative Commons, telif hakları konusunda esneklik sağlamayı amaçlayan, eser sahibinin haklarını koruyarak, eserlerin paylaşımını kolaylaştırıcı modeller sunan, kar amacı gütmeyen bir organizasyondur. Bu organizasyona dahil olan eserler, kaynağı belirtmek ön şartıyla belirli kısıtlamalar göz önünde bulundurularak kullanılabilir.

Bilişim dünyasında yazılımları lisanslarına göre, özgür yazılımlar ve ticari yazılımlar olmak üzere ikiye ayırabiliriz. Özgür yazılım dünyasına ait GPL'ye (General Public Licence - Genel Kamu Lisansı) sahip yazılımlar ücretsiz kullanılabilirken, ticari faaliyet gösteren firmaların ürettiği yazılımların lisanslarıysa çoğunlukla yüksek bedeller karşılığında alınabilmektedir.

Lisanssız yazılım kullanmanın etik uygunsuzluk yanında hukuki yaptırımları ve teknik sakıncaları da vardır.

Uluslararası Bilgisayar Etik Enstitüsüne göre bilişim teknolojilerinin doğru bir şekilde kullanılabilmesi için aşağıda belirtilen 10 kurala uyulması gerekmektedir.

1. Bilişim teknolojilerini başkalarına zarar vermek için kullanmamalısınız.
2. Başkalarının bilişim teknolojisi aracılığı ile oluşturduğu çalışmaları karıştırmamalısınız.
3. Başkasına ait olan verileri incelememelisiniz.
4. Bilişim teknolojilerini hırsızlık yapmak için kullanmamalısınız.
5. Bilişim teknolojilerini yalancı şahitlik yapmak için kullanmamalısınız.
6. Lisanssız ya da kırılmış/kopyalanmış yazılımları kullanmamalısınız.
7. Başkalarının bilişim teknolojilerini izinsiz kullanmamalısınız.
8. Başkalarının bilişim teknolojileri aracılığı ile elde ettiği çalışmalarını kendinize mal etmemelisiniz.
9. Yazdığınız programların ya da tasarladığınız sistemlerin sonuçlarını göz önünde bulundurmalısınız.
10. Bilişim teknolojilerini her zaman saygı kuralları çerçevesinde kullanmalı ve diğer insanlara saygı duymalısınız.

İNTERNET ETİĞİ

İnternet kullanımı ile ilgili olarak dikkat edilmesi gereken etik ilkeler; kişilik hakları, özel yaşamın gizliliği ve veri güvenliği gibi başlıklar altında incelenebilir. İnternet ortamında uyulması gereken etik kurallar aşağıda verilmiştir.

- 1: İnternet’te karşılaştığımız ancak yüzünü görmediğimiz, sesini duymadığımız kişilere saygı kuralları çerçevesinde davranmalıyız.
- 2: İnternet’i kullanırken her kültüre ve inanca saygılı olmak, yanlış anlaşılabilir davranışlardan kaçınmak gerektiği unutulmamalıdır.
- 3: Özellikle sosyal medya, sohbet ve forum alanlarındaki kişiler ile ağız dalaşı yapmaktan kaçınmalı, başka insanları rahatsız etmeden yazışmaya özen göstermeliyiz.
- 4: Sürekli olarak büyük harfler ile yazışmanın İnternet ortamında bağırarak anlamına geldiği unutulmamalıdır.
- 5: İnsanların özel hayatına karşı saygı göstererek kişilerin sırlarının İnternet ortamında paylaşılmamasına dikkat edilmesi gerektiği unutulmamalıdır.
- 6: İnternet’te kaba ve küfürle bir dil kullanımından kaçınarak gerçek hayatta karşımadığımız insanlara söyleyemeyeceğimiz ya da yazamayacağımız bir dil kullanmamalıyız.
- 7: İnternet’i başkalarına zarar vermek ya da yasa dışı amaçlar için kullanmamalı ve başkalarının da bu amaçla kullanılmasına izin vermemeliyiz.
- 8: İnternet ortamında insanların kişilik haklarına özen göstererek onların paylaştığı bilginin izinsiz kullanımından kaçınmamız gerektiği de unutulmamalıdır.

Siber (dijital) Zorbalık: İnternet ortamında başkalarından kaynaklanan kötü davranışlara, İnternet etiğine uymayan davranışlara denir. Siber zorbalığa maruz kalınması durumunda yapılması gerekenler:

- 1: Zorbalık yapan hesaplara cevap vermeyiniz, onlarla tartışmaya girmeyiniz. İlk yapmanız gereken, zorbalık yapan hesabı engellemektir.
- 2: Bu hesapları, bulunduğunuz sosyal medya platformundaki “Bildir/Şikayet Et” bağlantısını kullanarak şikayet ediniz.
- 3: Size yönelik etik dışı davranışlar artarak ve ağırlaşarak devam ederse bunların ekran görüntülerini ve mesajları kaydediniz. Bu kanıtlarla birlikte ailenizin ya da rehber öğretmeninizin gözetiminde hukuki yollara başvurunuz.
- 4: Siber zorbalığa maruz kalan başka kişiler de olabilir. Böyle durumlarda bu kişilere ne yapmaları gerektiği konusunda yardımcı olabilir, kötü kullanım bildirimini siz de yapabilirsiniz.

BİLGİ GÜVENLİĞİ

Bilgi Güvenliği: Kişisel ya da kurumsal düzeyde bizim için büyük önem teşkil eden her tür bilgiye izin alınmadan ya da yetki verilmeden erişilmesi, bilginin ifşa edilmesi, kullanımı, değiştirilmesi, yok edilmesi gibi tehditlere karşı alınan tüm tedbirlere bilgi güvenliği denir.

Siber Suç: Bilişim teknolojileri kullanılarak gerçekleştirilen her tür yasa dışı işlemdir.

Siber Zorbalık: Bilgi ve iletişim teknolojilerini kullanarak bir birey ya da gruba, özel ya da tüzel bir kişiliğe karşı yapılan teknik ya da ilişkisel tarzda zarar verme davranışlarının tümüdür.

Siber Savaş: Farklı bir ülkenin bilgi sistemlerine veya iletişim altyapılarına yapılan planlı ve koordineli saldırılardır.

SAYISAL DÜNYADA KİMLİK VE PAROLA YÖNETİMİ

Parola: Bir hizmete erişebilmek için gerekli olan, kullanıcıya özel karakter dizisidir.

Şifre: Sanal ortamdaki verilerin gizliliğini sağlamak için veriyi belirli bir algoritma kullanarak dönüştüren yapıdır.

Parola, bilgi güvenliğinin en önemli ögesidir. Parolanın da ele geçirilmesi durumunda oluşacak zarar, bir evin anahtarını ele geçiren hırsızın sebep olacağı zarardan çok daha fazla olabilir. Parolanın kötü niyetli kişiler tarafından ele geçmesi durumunda:

- 1: Elde edilen bilgiler yetkisiz kişiler ile paylaşılabilir ya da şantaj amacıyla kullanılabilir.
- 2: Parolası ele geçirilen sistem başka bir bilişim sistemine saldırı amacıyla kullanılabilir.
- 3: Parola sahibinin saygınlığının zarar görmesine yol açabilecek eylemlerde bulunulabilir.
- 4: Ele geçirilen parola ile ekonomik kayba uğrayabilecek işlemler yapılabilir.
- 5: Parola sahibinin yasal yaptırım ile karşı karşıya kalmasına yol açabilir.

Güçlü bir parolanın belirlenmesi için aşağıdaki kurallar uygulanmalıdır.

- Parola, büyük/küçük harfler ile noktalama işaretleri ve özel karakterler içermelidir.
- Parola, -aksi belirtilmedikçe- en az sekiz karakter uzunluğunda olmalıdır.
- Parola, başkaları tarafından tahmin edilebilecek ardışık harfler yada sayılar içermemelidir.
- Her parola için bir kullanım ömrü belirleyerek belirli aralıklar ile yeni parola oluşturulması gerekir.

KİŞİSEL BİLGİSAYARLARDA VE AĞ ORTAMINDA BİLGİ GÜVENLİĞİ

Teknolojinin hızlı ilerleyişi ile birlikte gelen güvenlik riskleri ve insanların bu konudaki yetersiz farkındalıkları bilgisayar ve İnternet kullanımı sırasında pek çok tehlikenin ortaya çıkmasına neden olmaktadır.

Bilişim sistemlerinin çalışmasını bozan veya sistem içinden bilgi çalmayı amaçlayan Virüs, Solucan, Truva Atı ya da Casus yazılım gibi kötü niyetlerle hazırlanmış yazılım veya kod parçaları zararlı programlar olarak adlandırılır. Bu zararlı programlar:

- İşletim sisteminin ya da diğer programların çalışmasına engel olabilir.
- Sistemdeki dosyaları silebilir, değiştirebilir ya da yeni dosyalar ekleyebilir.
- Bilişim sisteminde bulunan verilerin ele geçirilmesine neden olabilir.
- Güvenlik açıkları oluşturabilir.
- Başka bilişim sistemlerine saldırı amacıyla kullanılabilir.
- Bilişim sisteminin, sahibinin izni dışında kullanımına neden olabilir.
- Sistem kaynaklarının izinsiz kullanımına neden olabilir.

Virüsler: Bulaştıkları bilgisayar sisteminde çalışarak sisteme ya da programlara zarar vermek amacıyla oluşturur. Virüsler bilgisayara e-posta, bellekler, İnternet üzerinden bulaşabilir. Bilgisayarın yavaşlaması, programların çalışmaması, dosyaların silinmesi, bozulması ya da yeni dosyaların eklenmesi virüs belirtisi olabilir.

Solucanlar: Kendi kendine çoğalan ve çalışabilen, bulaşmak için ağ bağlantılarını kullanan kötü niyetli programlardır. Sistem için gerekli olan dosyaları bozarak bilgisayarı büyük ölçüde yavaşlatabilir ya da programların çökmesine yol açabilir.

Truva Atları: Kötü niyetli programların çalışması için kullanıcının izin vermesi ya da kendi isteği ile kurması gerektiği için bunlara Truva Atı denmektedir. Truva Atları saldırganların bilişim sistemi üzerinde tam yetki ile istediklerini yapmalarına izin verir.

Casus Yazılımlar: İnternet'ten indirilerek bilgisayara bulaşan ve gerçekte başka bir amaç ile kullanılsa bile arka planda kullanıcıya ait bilgileri de elde etmeye çalışan programlardır.

Zararlı Programlara Karşı Alınacak Tedbirler:

- Bilgisayara antivirüs ve İnternet güvenlik programları kurularak bu programların sürekli güncel tutulmaları sağlanmalıdır.
- Tanınmayan/güvenilmeyen e-postalar ve ekleri kesinlikle açılmamalıdır.
- Ekinde şüpheli bir dosya olan e-postalar açılmamalıdır. Örneğin resim.jpg.exe isimli dosya bir resim dosyası gibi görünse de uzantısı exe olduğu için uygulama dosyasıdır.
- Zararlı içerik barındıran ya da tanınmayan web sitelerinden uzak durulmalıdır.
- Lisanssız ya da kırılmış programlar kullanılmamalıdır.
- Güvenilmeyen İnternet kaynaklarından dosya indirilmemelidir.

PROBLEM ÇÖZME KAVRAMLARI VE YAKLAŞIMLAR

Programlama: Bilgisayar programlarının yazılması, test edilmesi ve bakımının yapılması sürecine verilen isimdir.

Bilgi işlemsel düşünme: Bilgisayar biliminin kavramlarından yararlanarak problem çözme, sistem tasarlama ve insan davranışlarını anlama olarak tanımlanabilir.

Sorunu daha biçimsel bir şekilde yeniden ifade etmek, bir problemi anlamak için mükemmel bir tekniktir. Birçok programcı, diğer programcıları bir sorunu tartışmak için arar; sadece diğer programcıların yanıtı olabileceğini düşünür fakat aynı zamanda problemi yüksek sesle ifade etmek genellikle yeni ve yararlı düşünceleri tetikler.

PROBLEM ÇÖZME TEKNİKLERİ

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 biraz daha yaklaştıracak ipuçları elde etmenizi sağlayacaktır.

Problemi Tekrar İfade Edin: Önceki problemlerde de gördüğümüz üzere 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 anlamamanın ortaya çıkmasına ya da hedefin daha iyi anlaşılmasına neden olur.

Problemi Küçük Parçalara Ayırın: Verilen problemi adımlara ya da bölümlere ayırmak, çözümü kolaylaştırır. Bir problemi iki bölüme ayırdığımız düşünüldüğünde, her bir parçanın çözümünün tümünü çözmeye göre yarı yarıya kolaylaştığını düşünebiliriz.

Önce Bildiklerinizden Yola Çıkın: Programlama yaparken öncelikle bildiklerimiz ile başlamalı ve sonra yeni çözümler arayışına girmeliyiz. Problemi küçük parçalara bölerek çözebildiğiniz parçadan başlayınız.

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

Benzerlikleri 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 problemin belirli bir bölümü başka bir problemle benzerlik gösterebilir.

Deneme Yapın: Bazen bir problemi çözenin en kolay yolu denemek ve sonuçlarını gözlemlemektir. Bu, tahmin etmekten çok farklıdır. Bir çözümü tahminen öngörmek ile kodu yazarak denemek ve sonuçlarını incelemek çok farklı sonuçlar verir. Böylece problemi çözebilmek için gereken ipuçlarını elde edebilirsiniz.

Asla Vazgeçmeyin: Asla vazgeçmemek, kişisel bir özelliktir. Kararlılık, güven ve istek kaybolduğu zaman acık düşünemezsiniz, işlemler olması gerektiğinden uzun sürer ve gittikçe zorlaşır. Hatta öfke ve kızgınlığa bile dönüşebilir.

PROBLEM ÇÖZME ADIMLARI

- 1. Problemi Tanımlama:** Problemi çözmeye başlamadan önce problemin acık, anlaşılır ve çok doğru bir şekilde tanımlanmış olması gerekir. Problemin ne olduğunu bilemezseniz onu çöremezsiniz.
- 2. Problemi Anlama:** Çözüme doğru yol almadan önce problemi çok iyi anladığınızdan emin olmanız gerekir. Problemin neler içerdiğini ve kapsamını doğru anlamalısınız.
- 3. Problemin Çözümü İçin Farklı Yol ve Yöntemler Belirleme:** Problemin çözümü için olabildiğince farklı yol ve yöntem belirlemeli ve bu listenin, tüm olasılıkları içerdiğinden emin olmalısınız.
- 4. Farklı Çözüm Yolları Listesi İçerisinden En İyi Çözümü Seçme:** Bu adımda her bir çözümün olumlu ve olumsuz yönlerini ortaya koymalısınız.
- 5. Seçilen Çözüm Yolu ile Problemi Çözmek İçin Gerekli Yönergeleri Oluşturma:** Bu adımda numaralandırılmış ve adım adım yönergeler oluşturmanız gerekir.
- 6. 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.

Sayfa 47'deki Şekil 1.7: Problem çözme adımları görselini inceleyiniz.

PROBLEM TÜRLERİ

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 acık 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.

VERİ TÜRLERİ

Sayısal Veri: Sayısal veri, hesaplama işlemlerinde kullanılabilen tek veri türüdür. Pozitif ya da negatif tam sayılar ve reel sayılar kullanılabılır. Sayısal veriler; acılar, 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.

Veri Türü	Veri Seti	Örnek	Python'daki Veri Türü
Sayısal: Tam Sayı	Tüm sayılar	66578 -250	integer (int)
Sayısal: Reel Sayı	Tüm reel sayılar ve ondalık sayılar	-56,23 3,56	float

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. Karakterler sadece sayıdan oluşsa bile hesaplama işlemlerinde kullanılamaz.

Veri Türü	Veri Seti	Örnek	Python'daki Veri Türü
Karakter	Tüm rakamlar, harfler ve özel semboller	“A”, “Y”, “k”, “i”, “6”, “0”, “+”, “%”	string
Karakter Dizisi	Birden fazla karakterden oluşan kombinasyon	“Bilgisayar”, “532-5556633”	string

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.

Veri Türü	Veri Seti	Örnek	Python'daki Veri Türü
Mantıksal	Doğru/Yanlış True/False	2<3 => True 6>8 => False	Boolean (bool)

Kitabınızın 51.sayfasındaki örnekleri inceleyiniz.

Bilgisayar veriyi 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.

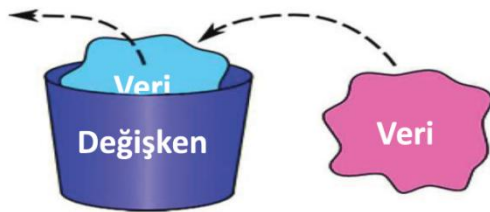
SABİT VE DEĞİŞKENLER

Sabit: 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.

Değişken: Sabitlerin 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 değişkenin içerisinde 50 değeri atanmış olabilir, program çalıştığı süre içerisinde bu değer değişebilir ancak değişkenin ismi hiçbir zaman değişmez.

Değişken isimlendirilirken dikkat edilmesi gerekenler:

1. Değişkene içerdiği değer ile tutarlı isimler veriniz.
2. Değişkenlere isim verirken boşluk kullanmayınız.
3. Değişkenlere isim verirken bir karakter ile başlayınız.
4. Matematiksel semboller kullanmamaya dikkat ediniz.
5. Bazı platformlar desteklemediği için Türkçe karakter kullanımı tavsiye edilmez.
6. Programlama dillerinde kullanılan komut isimleri değişken olarak kullanılamaz. Çok bilinenleri; if, for, while, else, do, int, vb.
7. 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. Örnek: tcKimlikNo
8. Özel karakterler değişken isimlerinde kullanılamaz (*,/, -,+, #,%,&,(,=,?,\$,[,,{ gibi...).



taksitSay = 9 (*int*)

adSoyad = “Ali Kara” (*string*)

değişken

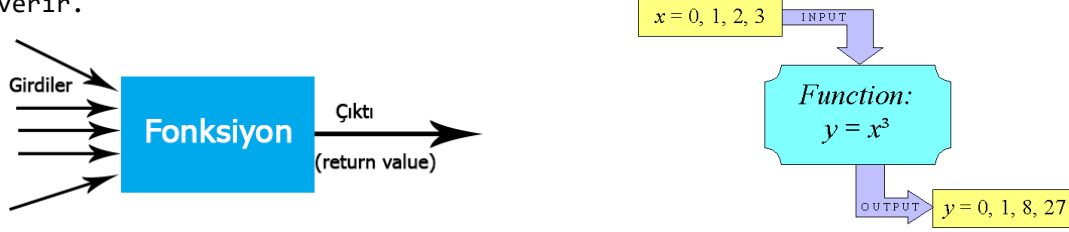
değer

? Sizce hayali bir program içerisinde aşağıdaki değerleri tutacak olan değişkenleri nasıl isimlendirebiliriz?

Hesap Numarası, Okul Numarası, Kitap Sayısı, Doğum Tarihi, Anne Kızlık Soyadı, 1.Yazılı Notu

FONKSİYONLAR

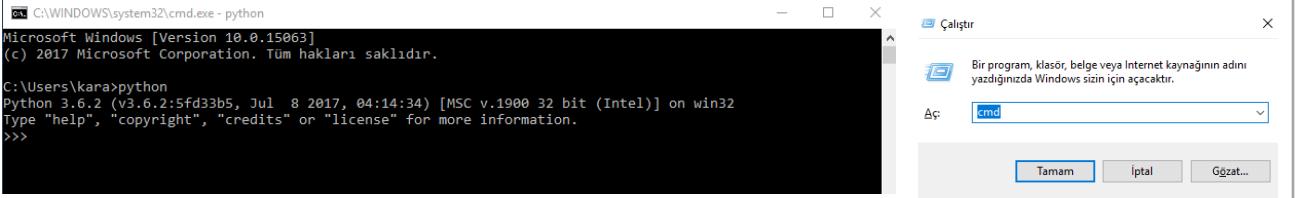
Fonksiyonlar, belirli işlemleri yürüten ve sonuçları döndüren bir işlem kümesidir. Her programlama dili, içerisinde kendine özgü fonksiyonlar barındırır. Bu fonksiyonlar kütüphanesi, programlama dili bilgisayara göre değişiklik gösterir. Ayrıca pek çok programlama dili, programcılarının kendi fonksiyonlarını yazmalarına da olanak verir.



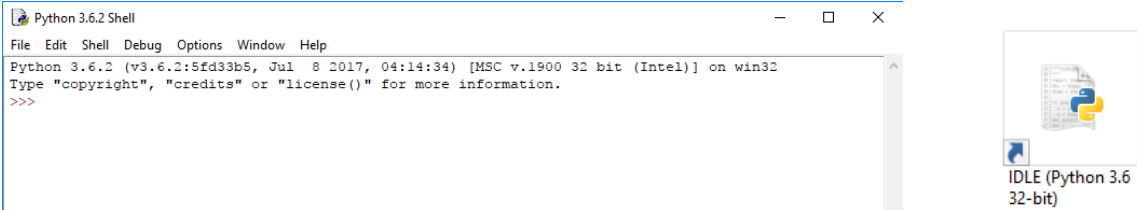
Fonksiyona gönderilen verilere **“parametre”** denir. Fonksiyonlar parametreleri değiştirmez ama işlemlerde kullanır. Örneğin karekök fonksiyonunu ele alalım. `Sqrt(N)`, gönderilen `N` değeri için karekök değeri hesaplamaktadır. `Sqrt` fonksiyonun ismi, `N` işlem yapılacak veri yani parametredir.

ETKİLEŞİMLİ KABUK (Interactive Shell)

Etkileşimli kabuk, bizim Python programlama dili ile ilişki kurabileceğimiz, yani onunla etkileşebileceğimiz bir üst katmandır. Etkileşimli kabuk, asıl programımız içinde kullanacağımız kodları deneme imkanı sunar bize. Burası bir nevi test alanı gibidir. Python kurulu bir bilgisayarda etkileşimli kabuğa erişmek için windows işletim sisteminde **başlat/çalıştır/cmd** yolunu izleyerek açılan komut satırına **python** yazarak python etkileşimli kabuğuna erişebiliriz.



Diğer bir yol olarak da, python kurulumuyla beraber gelen **IDLE** aracını çalıştırarak da python etkileşimli kabuğuna erişim sağlayabiliriz.



Bundan sonraki derslerimizde zaman zaman kodlarımızı denemek için etkileşimli kabuğu kullanacağız.

Sayfa 54’de bulunan örnek fonksiyonları ve çıktılarını inceleyiniz.

Aşağıdaki fonksiyonları etkileşimli kabuk üzerinde çalıştırıp sonuçlarını yazınız.

Aşağıdaki fonksiyonları etkileşimli kabuk üzerinde çalıştırıp sonuçlarını yazınız. >>> işaretlerini yazmayın :) Bu işaret etkileşimli kabuğun bizden yeni komut almaya hazır olduğu anlamına gelir.

```
>>> import math
>>> math.sqrt(81)
```

```
>>> isim = "Saime İnal Savi"
>>> len(isim)
```

```
>>> print("merhaba dünya")
```

```
>>> math.sqrt(abs(-144))
```

```
>>> abs(-40)
```

```
>>> math.cos(0)
```

```
>>> a = "5"
>>> type(a)
```

```
>>> a = "5"
>>> int(a)
```

Fonksiyonları iç içe de kullanabildiğimize dikkat edin.

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.

Operatörler; **matematiksel**, **mantıksal** ve **ilişkisel** operatörler olarak sınıflandırılabilir.

Matematiksel Operatörler

Operatör	Python'daki Sembolü	İşlem	Sonuç
Toplama	+	5.1+8.2	13.3
Çıkarma	-	8.5-5.0	3.5
Çarpma	*	4*3	12
Bölme	/	12/4	3
Modül Alma	%	12%3	0

Modül, bir sayının başka bir sayıya bölümünden **kalandır**.

$$\begin{array}{r} 3 \quad 5 \\ \hline 0 \quad 0 \\ \hline 3 \text{ kalan} \end{array} \quad \begin{array}{r} 33 \quad 5 \\ \hline 30 \quad 6 \\ \hline 3 \text{ kalan} \end{array} \quad \begin{array}{r} 48 \quad 5 \\ \hline 45 \quad 9 \\ \hline 3 \text{ kalan} \end{array}$$

```
>>> 3%5
```

```
3
```

```
>>> 33%5
```

```
3
```

```
>>> 48%5
```

```
3
```

```
>>> 12%3
```

```
0
```

Mantıksal Operatörler

Operatör	Python'daki Sembolü	İşlem	Sonuç
Eşittir	==	2==1	False
Büyüktür	>	2>1	True
Küçüktür	<	2<1	False
Küçük ya da eşittir	<=	2<=1	False
Büyük ya da eşittir	>=	2>=1	True
Eşit değildir	!=	2!=1	True

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

İlişkisel Operatörler

Operatör	Python'daki Sembolü	İşlem	Sonuç
Değil	NOT	not 1==1	False
Ve	AND	1==1 and 2==2 1==1 and 1==2	True False
Veya	OR	1==1 or 1==2 3==1 or 1==2	True False

AND operatörünün kullanıldığı bir ifadenin sonucunun True olabilmesi için ifadedeki tüm önermelerin sonucunun True olması gerekir.

OR operatörünün kullanıldığı bir ifadenin sonucunun True olabilmesi için ifadedeki önermelerden **birinin** True olması yeterlidir.

İŞLEM ÖNCELİĞİ

Matematiksel, mantıksal ve ilişkisel operatörlerin bir hiyerarşisi yani öncelikleri vardır. İşlemler, bu sıralamaya göre yapılmaz ise sonuç, beklendiği gibi çıkmayabilir. En içteki ayraçtan en dıştakine doğru işlem yapılmalı, ayraç içerisinde ise işlem önceliklerine dikkat edilmelidir.

() hiyerarşiyi sıralar, ayraç içerisindeki işlemler en içten en dışa doğru yapılmalıdır.

Matematiksel Operatörler

** Kuvvet (Üs)
% Mod
*, /
+, -

İlişkisel Operatörler

==, <, >, <=, >=, !=

Mantıksal Operatörler

NOT
AND
OR

İFADE ve EŞİTLİKLER

Şimdiye kadar yaptığımız işlemlerde hep **ifadeleri** kullandık. İfadeler bizlere sonuç döndürür ancak bu sonuçlar hafızada saklanmaz. İfadelerin sonuçlarını ileride program içerisinde yeniden kullanabilmek için **eşitliklerden** faydalanırız.

İFADELER	EŞİTLİKLER
A + B A ve B sayısal veridir. Sonuç sayısaldır ve hafızada korunmaz.	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 korunmaz.	C = A < B A, B ve C sayısal, karakter ya da dizi olabilir. Sonuç mantıksal değerdir ve C değişkenine atanarak korunur.
A OR B A ve B mantıksal veridir. Sonuç mantıksal veridir ve hafızada korunmaz.	C = A OR B A, B ve C mantıksal veridir. Sonuç mantıksal veridir ve C değişkenine atanarak korunur.

ÖDEV

Aşağıdaki ifade ve eşitlikleri etkileşimli kabuk üzerinde çalıştırıp, sonuçlarını yazınız.

```
>>> a = 5
>>> b = 3
>>> not a>b
```

```
>>> a = 3
>>> b = 2
>>> a>b or b>4
```

```
>>> a = 3
>>> b = 2
>>> a>b and b>4
```

```
>>> a = 5
>>> b = 3
>>> a*b < 20
```

```
>>> a = 5
>>> b = 3
>>> not a>b or b==5
```

```
>>> x = 3
>>> y = 4
>>> x*y > x**2
```

```
>>> a = 5
>>> b = 3
>>> not a>b or b==5
```

```
>>> a = 4
>>> b = 3
>>> a%b == 1
```

```
>>> a = 4
>>> b = 2
>>> a**b == b**a
```

```
>>> a = 4
>>> b = 2
>>> a**b != b**a
```

```
>>> kısaKenar = 4
>>> uzunKenar = 6
>>> alan = kısaKenar*uzunKenar
>>> alan
```

```
>>> x = 20
>>> y = 15
>>> z = (x*y)%2
>>> z == 0
```

```
>>> x = 3
>>> y = 4
>>> z = x*y
>>> z >= x**y or x<4
```

```
>>> x = 3
>>> y = 4
>>> (x*y)%5
```

```
>>> a = 4
>>> b = 2
>>> a!=b or 2<1
```

```
>>> a = 3
>>> b = 4
>>> a!=b and 2<1
```

```
>>> a = 3
>>> b = 4
>>> a!=b and not b<a
```

```
>>> x = 3
>>> y = 4
>>> (x**y)%5
```

BİLGİSAYAR İLE NASIL İLETİŞİM KURULUR?

Bilgisayarlar bizim konuştuğumuz dili bilemediğinden onlarla anlaşmamız için bizim onların konuştuğu dili öğrenmemiz gerekir. Bilgisayarın işletim sistemi, dili ve uygulamalarına ilişkin kurallara “**söz dizimi**” denir. Bir hata oluşursa buna “**yazılım hatası**”; hatayı bulup düzenleme işlemine ise “**hata ayıklama**” denir. Yazılım hataları bazen söz dizimi hatalarından bazen de mantık hatalarından kaynaklanabilir. Bu hatalar problem çözme sürecinde bulunarak düzeltilir. Programın hatasız çalışması ve doğru sonucu üretebilmesi için tüm hataların düzeltilmiş olması gerekir.

Problem çözme sürecini destekleyen bazı düzenleme araçları vardır. Bu araçları kullanmak; çözüme daha hızlı ulaşmak, en etkili programı yazmak, anlaşılır olmak ve süreci kolaylaştırmak için önemlidir.

1. Problem Analiz Çizelgesi,
2. Etkileşim Çizelgesi,
3. GSC (Girdi Süreç Çıktı) Çizelgesi,
4. Algoritmalar,
5. Akış Şemaları’dır.

1. PROBLEMİN ANALİZ ÇİZELGESİ

Çözümü düzenlemek için önce programın beklentilerini analiz etmek gerekir. Bunun için en iyi yol, problemi dört aşamada ele almaktır:

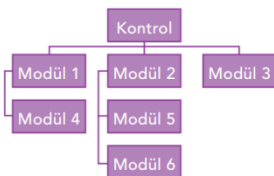
1. Eldeki veri
2. Beklenen sonuç
3. Problemin çözüm süreci
4. Çözüm seçenekleri

Bir örnek problem için problem analiz çizelgesinin nasıl olduğuna bir göz atalım: Sınav ve performans puanlarına göre ortalama hesaplama ve geçme kalma durumunun kontrolü:

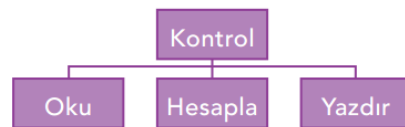
Eldeki Veri	Beklenen Sonuç
2 Yazılı ve 2 Performans Puanı	Geçme/Kalma Durumu
Problemin Çözüm Süreci	Çözüm Seçenekleri
-Ortalama = (Yazılı 1 + Yazılı 2 + Performans 1+ Performans 2)/4 -Geçme/Kalma Durumu= Eğer ortalama 50’den küçükse “Kaldı”, değilse “Geçti”	Yazılı ve performans puanlarını girecek değerler olarak tanımlama

2. ETKİLEŞİM ÇİZELGESİ GELİŞTİRME

Çözüme ulaşma yolunda ikinci adım, çözüm sürecini modüllere ayırmak ve süreçteki modüllerin birbiri ile etkileşimini görmek için modülleri birleştirmektir. Yönetisel etkileşim çizelgesi hazırlanırken yukarıdan aşağıya yaklaşım kullanılır. Tüm modülleri kontrol eden bir ana kontrol mekanizması dâhilinde süreç yukarıdan aşağıya doğru işler.



Şekil 1.9: Etkileşim çizelgesi



3. GSÇ (GİRDİ-SÜREÇ-ÇIKTI) ÇİZELGESİ

GSÇ (girdi-süreç-çıkı) çizelgesi problem analiz çizelgesindeki bilgiyi detaylandırır ve düzenler. GSÇ çizelgesi dört bölümden oluşur: **girdi**, **süreç**, **modül referansı** ve **çıkı**

Girdi	Süreç	Modül Referansı	Çıkı
Program için gerekli tüm veriler	Adım adım işlemler (Problem Analiz Çizelgesindeki 3 ve 4. Adımlar)	Etkileşim çizelgesindeki modüller	Tüm çıkı beklentileri (Problem Analiz Çizelgesindeki 1 ve 2. Adımlar)

Girdi	Süreç	Modül Referansı	Çıkı
Sınav ve Performans Puanları	Sınav puanlarını gir. Performans puanlarını gir. Puan ortalamasını hesapla. Ortalama>50 mi? Kontrol et Geçme kalma durumunu ekrana yazdır. Bitir.	Oku Oku Hesapla Karar Yazdır Kontrol	Geçti/Kaldı

4. ALGORİTMALAR

Yukarıdaki çizelgeler geliştirildikten sonraki adım, yapılacak işlemleri bilgisayarın anladığı dilde yazabilmektir. Bu yönergeler **“algoritma”** olarak adlandırılır.

“Sözde kod” algoritmaya çok benzer bir dildir ve bazen algoritma yerine kullanılabilir. Algoritmayı oluşturmak, bilgisayarda problem çözme sürecinin en zor bölümüdür. Modüller etkileşim çizelgesinden ve süreç GSÇ çizelgesinden alınır. Algoritmadaki işlem sayısı, programcının problemi çözme yoluna bağlıdır.










5. AKIŞ ŞEMALARI

Problem çözme sürecimiz, bilgisayarın iletişim kurma yöntemi ile şekillenir. **Algoritma**, bilgisayara hangi işlemi hangi sırada yapması gerektiğini söyleyen yönergeler bütünüdür.

Akış şeması ise algoritmanın görsel gösterimidir. Programcı, oluşturulan algoritmadan grafiksel gösterimler oluşturur. Akış şeması, program geliştirmeye başlamadan önceki son adımdır. Akış şemasında hatalar rahatlıkla görülüp düzeltilebilir. Akış şemalarını oluşturmak için kullanılan evrensel simgeler ve bu her bir simgenin anlamı vardır.

Akış şeması, bir problem çözümünün başlangıcından bitişine kadar olan süreci gösterir. Akış şeması içerisindeki her bir simge, algoritmadaki bir işlemi ifade eder. Genellikle işlemler tek yönlü olmasına rağmen karar kutularından iki farklı ok çıkar. Bir karar simgesinden çıkan ok, bazı işlemlerin tekrarlanmasını sağlayabilir; böylece bir “döngü” oluşur.

ALGORİTMA YÖNERGELERİ VE AKIŞ ŞEMASI SEMBOLLERİ

Simge	İşlev
	Başla/Bitir
	Giriş
	Atama/İşlem
	Denetim (Karar)
	Çıkış
	Döngü
	Akış Yönü
	Bağlaç
	Önceden Tanımlı İşlem/Fonksiyon

Şekil 1.10: Akış şeması sembolleri

Akış şemalarını oluştururken dikkat edilmesi gereken bazı noktalar şunlardır:

1. Yönergeler, simgelerin içine yazılmalıdır.
2. Hatırlatıcı bilgiler simgenin yanına yazılabilir. Böylece akış şeması ek açıklamalı bir şemaya dönüşür.
3. Bir akış şeması her zaman sayfanın başından başlar ve sonuna doğru gider. Eğer bir sayfaya sığmazsa bir ya da daha fazla bağlantı simgesi kullanılarak diğer sayfaya geçilebilir.
4. Akış şemasını çizmek için uygun yazılımlar kullanılırsa daha standart bir görünüm elde edilir.
5. Simgeler, içeriğindeki yazının rahatça okunabileceği kadar büyük yapılmalıdır.

HARİCİ VE DÂHİLİ DOKÜMANTASYON

İyi programcılar, kodları başkaları tarafından rahatça anlaşılabilirsin diye satırlar arasına açıklamalar yazarlar. Bu açıklamalar, diğer programcılar açısından büyük önem taşır çünkü kod üzerinde değişiklik yapılabilmesi için her bir satırın ya da fonksiyonun işlevinin anlaşılması gerekir. Bu şekilde, yazılıma ait **“dâhilî dokümantasyon”** oluşturulmuş olunur. Kod satırları haricinde yazılımın kullanımına ve teknik gereksinimlere ait bilgilerden oluşan “haricî dokümantasyon” hazırlanır. Bu bilgiler, diğer kullanıcılar tarafından ortaya çıkan problemleri çözmek için kullanılır.

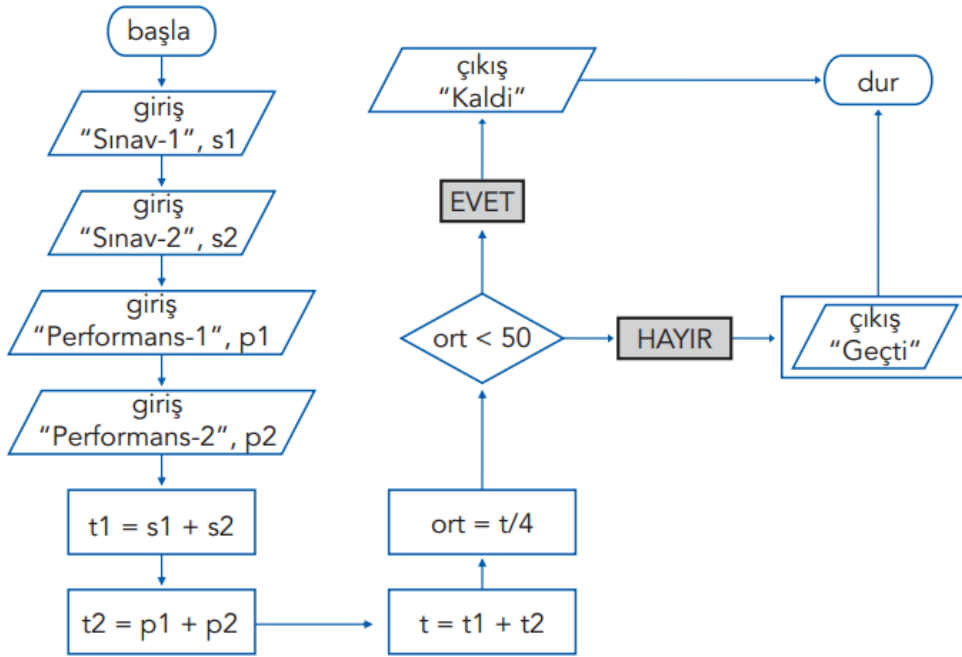
Python üzerinde # işaretinden sonraki ifadeler yorum satırı olarak değerlendirilir ve kodun çalışmasına etki etmez. Bu sayede programcılar kodlarının arasına açıklama satırları ekleyebilirler. Siz de etkileşimli kabukta bunu test ediniz.

ÇÖZÜMÜN PROGRAMLANMASI/KODLANMASI

Algoritmalar ve akış şemaları tamamlandıktan sonra istenilen bir programlama dili kullanılarak programın yazılması işlemine geçilir ki bu işleme “programlama” ya da “kodlama” adı verilir. Kodlama sonucunda programın ne kadar hatasız çalıştığı, algoritmanın etkililiğine bağlıdır.

Aşağıda iki yazılı ve iki performans puanı almış bir öğrencinin puan ortalamasını hesaplayarak, dersten geçip geçmediğini belirleyen akış şeması yer almaktadır.

Siz de benzer bir problemi çözüme kavuşturacak basamakları akış şemasıyla oluşturmayı deneyiniz.

**PROGRAMLAMA YAPISINA GİRİŞ**

Bir önceki bölümde anlatılan yaklaşımlar, problem çözümlerinin organize edilmesi için yardımcı olarak kullanılan araçlardı. Bu bölümden itibaren çözümleri bilgisayarın daha iyi anlayıp işleyebilmesi için kullanılan teknikler anlatılacaktır. Diğer bir ifade ile bu teknikler, algoritmayı oluşturan yönergeleri farklı biçimlerde yazmanıza olanak sağlayacaktır.

Göstergeler: Bilgisayarlar; problemleri çözmek, işlerimizi kolaylaştırmak, daha hızlı ve etkili çözümler üretmek için kullanılır. Gerçekten yeterli çözümler üretebilmek için aşağıdaki göstergeleri önemsemek gerekir.

1. Bütünü, her biri anlamlı işlemler içeren parçalara bölünüz, modülleri kullanınız.

2. Farklı satırlar arasında bağlantı kurmak yerine mantıksal yapıları kullanınız.

a) **Doğrusal yapı**, işlemleri sıra ile çalıştırır. Aşağıda, klavyeden girilen iki sınav puanının aritmetik ortalamasını hesaplayan yapı görülmektedir:

Algoritma	Akış Şeması	Sözde Kod
1. Başla. 2. Notları Oku. 3. Ortalamayı Hesapla. 4. Ortalamayı Yaz. 5. Bitir.	<pre> graph TD Start([Başla]) --> Input[/not1, not2/] Input --> Process[ort = (not1 + not2)/2] Process --> Output[ort] Output --> End([Bitir]) </pre>	1. Başla. 2. Oku <i>not1, not2</i> 3. <i>ort</i> = (not1 + not2)/2 4. Yaz <i>ort</i> 5. Bitir.

b) **Karar yapısı**, iki olasılıktan birini seçmek ve ona göre devam etmek için kullanılır. Sayfa 19'da yaptığımız akış şeması buna örnektir.

c) **Döngüsel yapı**, bir dizi işlemi tekrarlamak için kullanılır. 20 öğrencilik bir sınıfın bir dersten aldığı 2 not üzerine sınıf ortalamasını hesaplayan yapıya ait algoritma ve akış şemasını ders kitabının 67.sayfasından inceleyiniz.

d) **Durumsal yapı** ise belirli bir duruma göre farklı işlemlerin yapılmasına olanak sağlar.

Klavyeden girilen sayıya göre haftanın gününü yazan yapıya ait algoritma ve akış şemasını ders kitabının 68.sayfasından inceleyiniz.

3. Tekrarlayan işlemlerin tekrar tekrar yazılmasını önlemek için modüler yapı kullanınız.

4. Okunabilirliği ve anlaşılabilirliği artırmak için anlamlı değişken isimleri seçiniz ve çok iyi dokümantasyon hazırlayınız.

Ders kitabınızın 69, 70 ve 71. sayfalarındaki soruları arkadaşlarınızla birlikte çözmeye çalışın.

MODÜLLER VE İŞLEVLERİ

Bir yazarın, kitabını yazmaya başlamadan önce konuyu ve bölümleri düşünmesi, bir aşçının menüyü hazırlamaya başlamadan önce yemek türlerini, malzemeleri ve miktarları düşünmesi gibi bir programcı da programı yazmaya başlamadan önce detaylı bir biçimde problemi irdelemeli ve işlemleri gruplandırmalıdır. Ne zaman modüller etkileşim çizelgesinde doğru sıralanmış ise programcı her bir modül için kodu yazmaya başlayabilir.

Modülleri oluştururken aşağıdaki noktalara dikkat edilmesi önerilir:

1. Her bir modül başlar, işlemleri yapar ve biter. Süreç içerisinde modüller arasında dallanma olmaz.
2. Her bir modülün tek bir işlevi vardır: yazdırma, karekök bulma, büyük harfe çevirme vb.
3. Her modül rahat anlaşılabilir ve kolayca güncellenebilecek kadar kısa olmalıdır.
4. Modülün uzunluğu işlevine ve yönerge sayısına göre değişebilir.
5. Modüller süreç akışlarını kontrol etmek için oluşturulur.

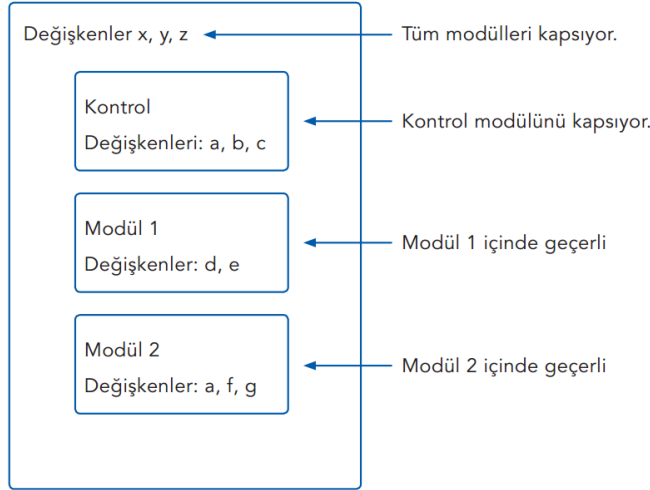
BAĞLILIK VE BİRLEŞİM

Modüller hem farklı işlemleri yürütecek kadar birbirinden bağımsız olmalı hem de aynı veriler ile çalışacak kadar birleşik olmalıdır. Birbirine zıt bu iki kavram bağıllık ve birleşim olarak adlandırılır.

YEREL VE GLOBAL DEĞİŞKENLER

Yerel ve global değişken kavramı tüm programlama dilleri için çok önemli kavramlardır. Programcılar, yerel ve global değişkenleri bağıllık ve yapışkanlık oluşturmak amacıyla kullanırlar. Bir modül içinde tanımlanmış değişkenler “yerel”; modüller dışında program genelinde kullanılmak üzere tanımlanmış değişkenler ise “global” değişkenler olarak adlandırılır.

Yerel değişkenler, yalnızca tanımlandıkları modül içerisinde kullanılabilir. Global olarak tanımlanan değişkenler ise bütün modüller tarafından tanınır.



PARAMETRELER

“Parametreler” bir modülden diğerine geçen yerel değişkenlerdir. Modüller arasındaki iletişimi sağlar. Modül adından sonra araç içerisinde belirtilerek kullanılırlar: Oku (a, b, c) gibi.

DÖNEN DEĞERLER

Dönen değer, fonksiyon sonucudur. Bu işlem, fonksiyonu adı ile çağırarak gerçekleşir. İşlem sonucundaki değer, geçici olarak ilgili değişkene atanır. Fonksiyon, çalışmasını bitirdiğinde artık o isme atanmış bir değer bulunmaz çünkü bu değer, çağırılan modüle geri dönmüştür. Aşağıdaki şekilde bu akış görülmektedir

$$\text{Sonuc} = 6 + \text{Kare}(2)$$

$\underbrace{\hspace{1.5cm}}_4$
 $\underbrace{\hspace{2.5cm}}_{10}$

$\text{Kare}(X) \ X=2 \downarrow$
 $\text{Kare}(2) = 4$

Algoritma	Akış Şeması	Sözde Kod
1. Başla. 2. Kare(2) işleminden dönen değer ile 6'yı topla. 3. Sonucu yaz. 4. Bitir. 1. Kare modülünü oluştur. 2. Başla. 3. Klavyeden girilen değeri kendisiyle çarp. 4. Sonucu döndür. 5. Bitir.		1. Başla 2. sonuc = 6 + kare(2); 3. Yaz sonuc; 4. Bitir 1. Modul kare(x); 2. Başla 3. s=x*x; 4. return s;

DOĞRUSAL MANTIK YAPISI

Bilgisayara birbiri ardına algoritmanın başından sonuna kadar sırası ile işlemesi gereken komutları veren bir programcı, bu yaklaşımı kullanıyor demektir.

Sayfa 65'de bulunan problemin çözümünde kullanılan algoritma buna örnektir.

KARAR MANTIK YAPISI İLE PROBLEM ÇÖZME

Karar yapıları, bilgisayara iki ya da daha fazla seçenek arasından seçim yapmak hakkı tanıyan önemli ve güçlü bir mantık yapısıdır. Eğer karar yapıları olmasaydı bilgisayarlar hızlı bir hesap makinesi olmanın ötesine gidemezdi. Karar yapıları, insanın düşünme tarzına çok uygun olduğu için anlaşılması son derece kolaydır.

Karar mantık yapısı, if-elif-else (eğer-koşul sağlanırsa-x, değilse y) yönergesini kullanır. Bu durumda, eğer bir koşul doğru ise belli yönergeler; değilse farklı yönergeler çalıştırılabilir. "elif" kısmı kullanılmak zorunda değildir; bazen bu durumlarda hiçbir yönerge olmayabilir.

Koşullar;

1. Mantıksal bir ifade (AND (VE), OR (YA DA) veya NOT (DEĞİL))
2. İlişkisel operatörleri kullanan bir ifade (<, <=, >=, =),
3. Sonucu doğru ya da yanlış çıkan mantıksal bir değişken,
4. Bu üç seçeneğin birleşiminden oluşan bir ifade olabilir.

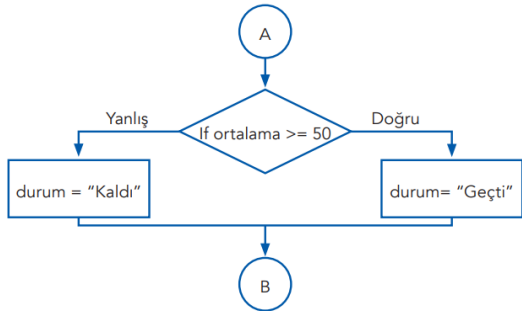
```

1 yas = int(input("Yaşınız: "))
2 if yas == 18:
3     print("18 iyidir!")
4 elif yas < 0:
5     print("Yok canım, daha neler!...")
6 elif yas < 18:
7     print("Genç bir kardeşimisin!")
8 else:
9     print("Eh, artık yaş yavaş yavaş kemale eriyor!")

```

TEK KOŞULLU YAPILAR

Koşulun sağlanıp sağlanmaması durumuna göre programın akışı değişir ve program, karara uygun yönergelerle çalışmaya devam eder. Hiçbir zaman üçüncü bir seçenek olamaz çünkü karar sembolünden yalnızca iki olasılık çıkabilir. Diğer bir ifade ile belirtilen durum ya doğrudur ya da yanlıştır.

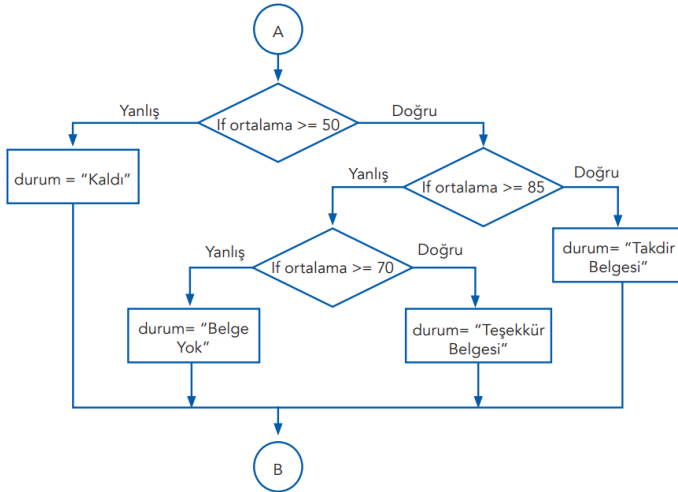


```

1 ortalama = int(input("ortalamanızı yazınız: "))
2 if ortalama >=50:
3     print("geçti")
4 else:
5     print("kaldı")
  
```

İÇ İÇE KARAR YAPILARI

Çoklu karar yapıları içeren algoritmalarda eğer koşullarını iç içe yazmamız gerekebilir. Aşağıdaki algoritma, öğrencinin puan ortalamasına göre Geçme/Kalma durumunu kontrol ettikten sonra, geçiyorsa öğrencinin belge alma durumunu belirlemektedir. Bu durumda bu örneğin algoritmasını aşağıdaki biçimde düzenleyebiliriz.



```

1 ortalama = int(input("ortalamanızı yazınız: "))
2 if ortalama >=50:
3     if ortalama >=85:
4         print("Takdir Belgesi")
5     elif ortalama >=70:
6         print("Teşekkür Belgesi")
7     else:
8         print("Geçti Ama Belge Yok")
9 else:
10    print("Kaldı")
  
```

DÜZ MANTIK KULLANIMI

Düz mantık ile çalışan kararlarda bütün koşullar test edilir. Bir koşulun test edilmesi, "Doğru" ya da "Yanlış" sonuç elde etmek için durumun işlenmesidir. Düz mantık çözümlerin içinde en yetersizi, çözüm olarak nitelendirilebilir çünkü bütün koşulların test edilmesi, programın çalışmasını da uzatır. **Örnek:** Tiyatro bileti alırken bilet fiyatı yaşa göre değişmektedir. Yaşı 18'den küçük olanlar için bilet ücreti 15 TL; yaşı 18'den büyük ve 65'ten küçük olanlar için 20 TL ve yaşı 65'ten büyük olanlar için 10 TL olarak belirlenmiştir. Bu durumda tiyatro seyircilerinin yaşlarına göre bilet almalarına olanak sağlayan bir çözüm geliştirmemiz gerekir. Bu problemin düz mantık yapısı ile şu şekildedir.

(sayfa 88'deki akış şemasını inceleyiniz)

```

1 yas = int(input("Yaşınız: "))
2 if yas <= 18:
3     ucret = 15
4 if yas > 18 and yas < 65:
5     ucret = 20
6 if yas > 65:
7     ucret = 10
8 print(ucret, "TL")
  
```

POZİTİF MANTIK KULLANIMI

Düşünme biçimimize en çok benzeyen yapı olması nedeni ile pozitif mantık kullanımı en kolay yapıdır. Pozitif mantık her zaman iç içe If/Elif/Else yapısını kullanır. Bu yapı; kullanıldığında genellikle bilgisayardan, koşulun doğru olması durumunda işlem yapması, yanlış olması durumunda farklı bir karar vermesi beklenir. Böylece daha az adımda karar verilebilir. Bir önceki problemin bu yaklaşım ile çözümünü inceleyelim.

(sayfa 89'daki akış şemasını inceleyiniz)

```
1 yas = int(input("Yaşınız: "))
2 if yas <= 18:
3     ucret = 15
4 else:
5     if yas<65:
6         ucret = 20
7     else:
8         ucret =10
9 print(ucret,"TL")
```

NEGATİF MANTIK KULLANIMI

Genellikle tersten düşünmediğimiz için negatif mantık yapısı, kurgusu, programcılara en zor gelen yapıdır. Negatif mantık kullanıldığında bilgisayardan, koşulun doğru olması durumunda farklı yönergeleri takip etmesi beklenir. Negatif mantık kullanmak, kontrol edilecek koşul sayısını azalttığından programı daha anlaşılır kılarak geliştirir.

(sayfa 90'daki akış şemasını inceleyiniz)

```
1 yas = int(input("Yaşınız: "))
2 if yas > 18:
3     if yas<65:
4         ucret = 20
5     else:
6         ucret = 10
7 else:
8     ucret =15
9 print(ucret,"TL")
```

HANGİ MANTIK YAPISI

Bir problemi çözmek için hangi karar yapısını seçeceğimize nasıl karar vereceğiz? Bunun en kolay yolu her 3 yapı için çözümü yazmak ve bu çözümler içinden en hızlı, kolay algılanan ve en az koşulla çalışanı seçmektir. Her zaman aynı yapıyı kullanmak ya da problemten istenildiği sıradaki yönergeleri kullanarak çözüm üretmek, sıkça başvurulan yollardır ancak bu yaklaşımlar her zaman en etkili çözüm ile sonuçlanmayabilir.

(sayfa 92'deki akış şemalarını inceleyiniz)

KARAR TABLOLARI

Karar tabloları, problemi birlikte çözdüğünüz kişi ile iletişim sağlamak ve süreci anlaşılır kılmak için çok kullanışlı bir araçtır. Karar tablosu, problem çözme mantığını tablo biçiminde gösteren bir araçtır. Akış şemalarının alternatifi de olabilir.

DÖNGÜ MANTIK YAPISI

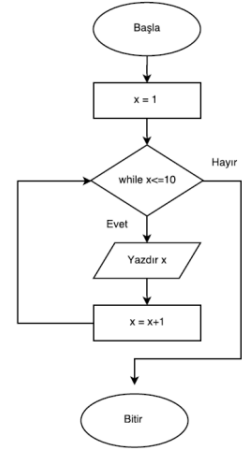
Problem çözüm sürecinde kullanılacak üçüncü bir karar yapısı, döngü yapısıdır. Bu yapı, tekrarlayan bir yapıdır. Çoğu problem, aynı işlemi farklı verileri kullanarak tekrar etmeyi gerektirir. Bu yüzden bu yapı son derece önemlidir.

WHILE Döngü Yapısı

While döngü yapısında şart sağlandığı sürece döngü tekrar eder.

```
1 x=1
2 while x<=10:
3     print(x, '. Tekrar')
4     x+=1
```

```
1 . Tekrar
2 . Tekrar
3 . Tekrar
4 . Tekrar
5 . Tekrar
6 . Tekrar
7 . Tekrar
8 . Tekrar
9 . Tekrar
10 . Tekrar
```



Arttırma/Azaltma: Döngüleri istenilen miktarda tekrar ettirebilmek için sayaçlardan faydalanabiliriz.

$x = x+1$ veya $x+=1$

$x = x-1$ veya $x-=1$

Biriktirme: Biriktirme işlemi arttırma işlemi ile çok benzerdir ancak her seferinde toplama ya da sonuca eklenen değer sabit olmayabilir. Biriktirme işlemi için kullanılan yönerge şu şekildedir:

toplam = toplam + degisken

FOR Döngü Yapısı

for deyimi ve for deyimi kullanılarak oluşturulacak döngü yapısı, işlemlerin tekrar sayısının önceden belli olduğu durumlarda kullanılır. Python'da genellikle **range()** fonksiyonu ile birlikte kullanılır.

range: Bu fonksiyonu belli bir aralıktaki sayıları listelemek için kullanıyoruz.

Örneğin, 0 ile 10 **arası** sayıların listesini almak istersek şöyle bir komut yazabiliriz:

```
>>> x = range(0, 10)
```

Ancak burada dikkat etmemiz gereken bir özellik var: Bu fonksiyon aslında doğrudan herhangi bir sayı listesi oluşturmaz. Yukarıda x değişkenine atadığımız komutu ekrana yazdırırsak bunu daha net görebilirsiniz:

```
>>> print(x)
range(0, 10)
```

Tipini yazdıralım.

```
>>> type(x) <class 'range'>
```

Gördüğümüz gibi, elimizdeki şey aslında bir sayı listesi değil, bir **'range' (aralık) nesnesidir**. Biz bu nesneyi istersek başka veri tiplerine dönüştürebiliriz. Mesela bunu bir listeye dönüştürelim:

```
>>> list(x) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Esasında, yukarıda nasıl kullanılacağına dair bazı örnekler verdiğimiz bu range() fonksiyonunu temel olarak şu şekilde formüle edebiliriz:

range(başlangıç_değer, bitiş_değeri, atlama_değeri)

range fonksiyonunu öğrendiğimize göre for döngü yapısını artık daha işlevsel kullanabiliriz.

FOR Döngü Yapısı

Örnek: Bir karakter dizisi içerisindeki elemanların her birini for döngü yapısı ile ekrana yazdıralım.

```
1 ornek_degisken = "Merhaba"
2 for i in ornek_degisken:
3     print(i)
4 print("Bu kısmın döngüye dahil olmadığına dikkat edin.")
```

```
M
e
r
h
a
b
a
Bu kısmın döngüye dahil olmadığına dikkat edin.
```

for ile **in** arasında kullandığımız **i** değişkeni, döngü içerisinde o esnada üzerinde gezindiğimiz elemanı ifade eder. Bunu istediğiniz şekilde isimlendirebilirsiniz. **for** döngüsü ile sadece karakter dizisi içerisinde değil, ileride göreceğimiz liste ve tuple gibi farklı veri tipindeki değişkenler üzerinde de gezinebiliriz.

Örnek: **range()** fonksiyonu kullanarak bir örnek yapalım.

```
for i in range(1,5):
    print(i, ". Tekrar")
```

! **range()** fonksiyonun 3.parametresi olan **atlama değerini** belirtmediğimizde bunun varsayılan olarak 1 kabul edildiğine dikkat edin.

! **range()** fonksiyonun 2.parametresi olan **bitiş değerinin** 1 eksiği kadar döngünün tekrar ettiğine dikkat edin.

Ders kitabının 103. sayfasındaki örnek probleme ait algoritma ve akış şemasını inceleyiniz.

break: Python'da **break** özel bir deyimdir. Bu deyim yardımıyla, devam eden bir döngüyü kesintiye uğratabiliriz. Bu deyimin kullanıldığı basit bir örnek verelim: 5 defa tekrar etmesi gereken döngüyü 2. tekrarda sonlandıralım. Programlama ünitesine geçtiğimizde daha detaylı inceleyeceğiz.

```
for i in range(1,6):
    print(i, ". Tekrar")
    if i==2:
        break
```

continue: **continue** deyimini de, aynı **break** gibi, sadece bir döngü içinde ve bir **if** şartı altında anlam ifade eder. **continue** döngü blokunun işlemlerini yarıda keser ve başa döner. **break**'den farkı, programın döngünün dışına çıkmaması, ama döngünün başına dönmesi ve tekrar başlatmasıdır. Bu arada döngü şartının doğru olup olmadığı da kontrol edilir.

5 defa tekrar etmesi gereken bir döngü yapısı içerisinde **continue** deyiminden faydalanarak tekrarlardan birisinin işleyişini kırıp, döngünün başa dönmesini sağlayan bir örnek yapalım.

```
for i in range(1,6):
    if i==2:
        continue
    print(i, ". Tekrar")
```

1.DÖNEM 1.YAZILI SINAVI (HEPİNİZE BAŞARILAR DİLERİM)

Otomatik Sayaç Döngüsü

While döngüsünü anlatırken döngümüzün tekrar sayısını kontrol edebilmek için kendimiz bir sayac değişkeni oluşturmuş ve bu değişkeni döngü içerisinde artırma/azaltma yoluyla kontrol etmiştik. For döngüsünde de range() fonksiyonundan faydalanarak döngümüzün tekrar sayısını kontrol edebilmiş ve break deyimi ile döngümüzün işleyişini sonlandırabilmiştik. Programlama ünitesinde bu konuları daha ayrıntılı bir şekilde işleyeceğiz.

Sayfa 103'deki örnek probleme ait algoritma ve akış şemasını inceleyiniz.

İç İçe Döngüler

Karar yapılarında olduğu gibi döngüler de iç içe kullanılabilir. İç içe döngüler, tekrarlayan bir dizi işlem yine tekrar edilmek istendiğinde kullanılır. İçteki döngülerin dıştaki döngülerle aynı döngü yapısında olması gerekmez. Dıştaki döngü while yapısında iken içteki döngü for yapısında ya da tam tersi biçimde olabilir. Ancak içteki ve dıştaki döngüyü kontrol eden değişkenin farklı olması gerekir.

Programlama ünitesinde iç içe döngüleri daha ayrıntılı bir şekilde işleyeceğiz.

II. BÖLÜM – PYTHON İLE PROGRAMLAMA

Yazılım Geliştirme Süreci

Yazılım geliştirme süreci şu şekilde işler;

- Programcı programlama bir dili kullanarak kodları oluşturur,
- Yazılan kod bütünü, hata ayıklayıcı (debugger) kullanılarak hatalara karşı denetlenir,
- Hataları giderilmiş kodlar, derleyici (compiler) kullanılarak bilgisayarın yorumlayabileceği elektriksel sinyallere dönüştürülür.

Bu süreç sonunda bilgisayar, elektriksel sinyalleri yorumlayarak **komutların gereğini yapar.**

Yazılım

Bilgisayarı belirli işlevleri yerine getirmek üzere yöneten, bilgisayara ne yapacağını söyleyen, kodlanmış komutlar dizisidir. (Bakınız: Ders Notları Sayfa 1)

Yazılım Geliştirme Ortamları

Programlama dillerini kullanabilmek için önceden her birinin kendine özgü olan yazılım geliştirme ortamlarının hazırlanması gerekir. Bu ortamlar programlama dilinin genel mantığının sistematik bir şekilde oturtulduğu, genel işleyiş içerisinde programcıya gerekli araçları sunan bir platformdur. Yazılım geliştirme ortamlarının genel amacı kullanıcıya üretkenlik sağlamaktır. Bu şekilde kullanıcı bir takım kolaylıklar elde ederken bazı kontrolleri bu platformlara bırakır.

Editörler

Programcının kaynak kodu yazmasını, test etmesini ve kaydetmesini sağlayan ortamlara kodlama editörleri denir. Çoğu editör, renklendirme desteği sunarak dilin özelliklerini ortaya çıkarır ve programcının üretkenliğinin artmasını destekler.

Python kodlamak için kullanabileceğiniz bir kaç editör:

[IDLE](#) (Python ile birlikte kurulu gelen editör)

[Geany](#) (Küçük boyutlu ve oldukça kullanışlı bir python editörü)

[Wing IDE](#) (Yaygın olarak kullanılan bir python editörü)

[PyCharm](#) (Daha çok profesyonel geliştiriciler tarafından tercih edilen gelişmiş bir python editörü)

[repl.it](#) (online olarak python kodlarınızı yazıp test edebileceğiniz bir platform)

Derleyiciler

Derleyiciler, kaynak kodları hedef koda dönüştürür. Hedef kod, belirli bir platform ya da gömülü bir araç için makine dili olabilir.

Derleyici yapısının grafiksel gösterimi için sayfa 111, şekil 2.1'i inceleyiniz.

Yorumlayıcılar

Yorumlayıcılar da derleyiciler gibi üst düzey kaynak kodu hedef koda (genellikle makine kodu) çevirir ancak derleyicilerden farklı çalışır. Kodu satır satır veya bloklar halinde çalıştırıp sırası gelmeyen satırları hiç çalıştırmayan dolayısıyla bu satırlardaki hataları hiçbir zaman göremeyen ve kodun bütününe ait iyileştirmeleri yapamayan çeviricilere yorumlayıcı denir. Derleyiciler, yorumlayıcılara göre daha hızlıdır. Çünkü yorumlayıcılar ilk kod satırından son kod satırına kadar her satırını teker teker yorumlar ve kodun karşılığındaki işlemi gerçekleştirir. Derleyiciler kodların tamamını bilgisayar diline çevirir. Eğer hata varsa, tüm hataları programcıya bildirir. Ancak yorumlayıcılar karşısına ilk çıkan hatayı bildirmektedir, ilk hata çözülene kadar diğer hataları bulamaz çünkü satır satır işlem yapmaktadır.

Yorumlayıcı yapısının grafiksel gösterimi için sayfa 111, şekil 2.2'yi inceleyiniz.

Hata Ayıklayıcılar

Hata ayıklayıcılar, programcının bir programdaki olası hataları bulmasına ve düzeltmesine olanak sağlayarak programın doğru çalışması için yardımcı olur. Hata ayıklayıcı programlar ile programın hangi satırlarında hata olduğu belirlenir. Programcı, oluşan hatanın detayına göre neyin yanlış gittiğini anlayıp, hatayı giderir.

Neden Python?

Python, öğrenmesi kolay, tamamen özgür ve ücretsiz bir programlama dilidir. Nesnelere dayalı bir dil olup okunabilirliği yüksektir. Python'un dili başka programlama dilleri ile kıyaslandığında, bunun daha az kod ile işlemleri yapmasının mümkün olduğu görülecektir. Python, bütün işletim sistemleri ile uyum içerisinde çalışmaktadır.

Python Sürümleri

Python programlama dilinin Kasım 2017 itibarıyla en güncel sürümü Python 3.6.3'dür. Ders kitabında ve bu notlarda yer alan örnekler 3.x sürümlerinde çalışan uygulamalardan oluşmaktadır. Ancak başka kaynaklarda yer alan birçok örneğin daha önceki sürümlerde (Python 2.x) yazıldığını görmeniz mümkün olabilir. Bu sürümler arasında söz dizimsel farklılıklar mevcuttur. Farklı kaynaklardan bulduğunuz örnekler yeni sürümlerde çalışmayabilir. Bu nedenle kitapta yer alan söz dizimi kurallarına göre kodların değiştirilmesi gerekmektedir.

Hafta: 10 / Ders: 1**DEĞERLER VE DEĞİŞKENLER****Tam Sayı ve Karakter Dizileri:**

Herhangi bir sayı, sayısal değerdir. Örneğin matematikte 4 sayısı tam sayı olarak ifade edilir. Tam sayılar pozitif, negatif ya da sıfır değeri alabilir. Kesirli değerleri içermez. Örneğin 99, 105, 0, -56 ve 7896 birer tam sayıdır. Daha önceden de bahsetmiş olduğumuz gibi tam sayıların veri türü int (integer) 'dır.

Üzerinde matematiksel işlem yapılamayan, tek veya çift tırnak içerisine aldığımız metinsel verilere de karakter dizisi denilmektedir. Bunlara ait veri türüne de str (string) dediğimizi hatırlayınız.

Etkileşimli kabukta aşağıdaki işlemleri yapıp sonuçlarını inceleyiniz.

```
>>> a=4
>>> type(a)
<class 'int'>
```

```
>>> b="4"
>>> type(b)
<class 'str'>
```

```
>>> int(b)
4
```

```
>>> c="mehmet"
>>> int(c)
```

Traceback (most recent call last):

```
File "<pyshell#8>", line 1, in <module>
    int(c)
```

ValueError: invalid literal for int() with base 10: 'mehmet'

Değişkenler ve Atama

Değişkenler, değerleri korumak için kullanılır. Bu değerler sayı, dizi gibi farklı biçimlerde olabilir. Örneğin;

```
x = 10
```

ifadesi bir atama satırıdır. Atama işlemi bir değeri bir değişken ile eşleştirir. Bu ifadedeki en önemli ayrıntı, atama (=) sembolüdür. Bu ifade ile 10 değeri, x değişkenine atanmaktadır. Bu noktada, x değişkeninin türü tam sayı olur çünkü atanan değer bir tam sayı değeridir. Bir değere birden fazla kez atama yapılabilir. Eğer bu sırada öncekinden farklı türdeki bir değer ataması yapılırsa değişkenin türü de değişir. Burada atama (=) sembolünün anlamı matematikte kullanıldığı şekliinden daha farklıdır. Matematikte bu sembol, eşitlik sağlar, bu yüzden bu sembolün sağ ve sol tarafında yer alan ifadelerin birbirine eşit olduğu anlamına gelir. Python dilinde ise atama (=) sembolünün sol tarafında yer alan ifade, sağ taraftaki ifadeyi üstlenir. Bu yüzden bu ifadeyi “5 değeri x değişkenine atandı.” ya da x’e 5 atandı.” şeklinde yorumlamak doğru olacaktır. Bir değere defalarca farklı değerler atayabiliriz. *Sayfa 121-122’yi inceleyiniz.*

Reel Sayılar

Pek çok hesaplamalı işlem, kesir parçası olan sayıları kullanır. Örneğin bir dairenin çevresini ya da alanını hesaplamak için π değerine ihtiyacımız vardır ve bu değer yaklaşık 3.14159 olarak ifade edilir. Python, bu şekilde noktalı sayılarla işlem yapar ve bu sayılara reel ya da gerçek sayı denir. Daha önceden de hatırlayacağınız üzere bu tür sayıların veri türüne **float** denilmektedir.

Bazı durumlarda float türündeki verilerimizi integer’a çevirmemiz gerekebilir. Bu durumda 2 fonksiyondan faydalanabiliriz. Bunlardan ilki olan `int(x)` fonksiyonu her zaman bir alt tam sayıya yuvarlarken, `round(x)` kendisine en yakın tam sayıya yuvarlar. `round()` fonksiyonunun 2. parametresi ile yuvarlama hassasiyetini belirleyebilirsiniz. *Sayfa 123.deki örnekleri inceleyiniz.*

```
>>> 28.71
28.71
>>> int(28.71)
28
>>> round(28.71)
29
>>> round(19.47)
19
>>> int(19.47)
19
```

3.İFADELER VE ARİTMETİK İŞLEMLER**Python’da Sık Kullanılan Aritmetik İkili Operatörler**

Ders notlarımızın 13. ve 14.sayfalarında python’da kullanılan operatörleri görmüştük. Bu operatörlerin sayısal veri türleri ve karakter dizisi veri türlerinde ne gibi sonuçlar döndürdüğünü incelemek için sayfa 127’deki tabloyu inceleyiniz.

Operatörler ve İşlem Önceliği

Ders notlarımızın 14.sayfasındaki işlem öncelikleri python’da da aynen geçerlidir.

Yorum Satırları

Python’da # karakteri ile başlayan satırlar python tarafından görmezden gelinerek kod olarak değerlendirilmez.

Sayfa 130’daki “aritmetik örnekler” başlığı altındaki örnekleri inceleyip uygulayınız.

4.KOŞULLU DURUMLAR

Boolean İfadesi

Bilgisayar bilimi temelde 0 ve 1 değerleri üzerine kurulmuştur; 0 değeri False(Yanlış), 1 değeri True(Doğru) demektir. Bu ifadelere Boolean (bool) İfadeleri denir. Doğru ve Yanlış değerleri korumak için kullanılan tipe bool adı verilmektedir. Sadece iki Boolean ifade değeri vardır:

- True (Doğru) (1)
- False (Yanlış) (0)

İlişkisel Operatörler

$x==y$ Eğer x ve y birbirine eşitse (matematiksel olarak) doğrudur, değilse yanlıştır.

$x<y$ Eğer x , y 'den küçükse doğrudur; değilse yanlıştır.

$x<=y$ Eğer x , y 'den küçük ya da eşitse doğrudur; değilse yanlıştır.

$x>y$ Eğer x , y 'den büyükse doğrudur; değilse yanlıştır.

$x>=y$ Eğer x , y 'den büyük ya da eşitse doğrudur; değilse yanlıştır.

$x!=y$ Eğer x , y 'den farklı ise (büyük ya da küçük) doğrudur; değilse yanlıştır.

if/else ifadesi

Belirli bir şartın sağlanması (sonucun True üretmesi) durumunda çalışmasını istediğimiz kod bloklarını **if** içerisine, şartın sağlanmaması durumunda çalışmasını istediğimiz kod bloklarını da **else** içerisine alarak programımızın belirli koşullara göre farklı işlemler yapmasını sağlayabiliriz. **if** ya da **else** bloklarına dahil etmek istediğimiz kodları 1 sekme (tab) içerden başlatmamız gerekir.

```
# kullanıcıdan sayı alıyoruz.
```

```
sayi = int(input("bir sayı gir:"))
```

```
# Eğer girilen sayının 2 ile bölümünden kalan 0 ise
```

```
if sayi%2==0:
```

```
    # ekrana "çift sayı" yazdırıyoruz.
```

```
    print("çift sayı")
```

```
# Değil ise
```

```
else:
```

```
    # ekrana "tek sayı" yazdırıyoruz.
```

```
    print("tek sayı")
```

if içerisindeki şartın sağlanmaması durumunda yeni şartlar tanımlayabilmek için **elif** ifadesini kullanabiliriz. Aşağıdaki örneği inceleyip, kendiniz yazarak çalıştırmaya çalışınız.

```
x = int(input("notunuzu girin: "))
```

```
if x<50:
```

```
    print("kaldı")
```

```
else:
```

```
    if x>=85:
```

```
        print("takdir belgesi")
```

```
    elif x>=70:
```

```
        print("Teşekkür Belgesi")
```

```
    else:
```

```
        print("Belgesiz Geçti")
```

Birleşik Boolean İfadesi

Daha önce öğrenmiş olduğumuz **and**, **or**, **not** mantıksal operatörlerini **if/elif** deyimleri içerisinde koşul tanımlayabilmek için kullanabiliriz.

Pass Deyimi

Pass ifadesi Python'da herhangi bir işlem yapmadan geçeceğimiz durumlarda kullanılır. Kısaca "Hiçbir şey yapmadan yola devam et!" anlamı katar.

```
if x == 2:
    print(x)
else:
    pass # x 2'ye eşit değilse hiçbir şey yapma
```

Çok Yönlü Karar İfadeleri

Basit if/else ifadesinde iki farklı koşul varken çok yönlü karar ifadelerinde daha fazla koşulun gerçekleşme durumuna göre işlem yapılır. Bunun için kod yazılırken iç içe if/else ifadeleri gerekir. Bununla ilgili derste yaptığımız örnekleri inceleyiniz.

Mantık Karmaşası

Python, çok karmaşık durum/koşul ifadelerini oluşturmak için gerekli araçları sağlar. Ancak önemli olan, mantık karmaşasına yol açmadan kullanabilmektir. Boolean ifadeleri and ve not ile birlikte kullanılmak istendiğinde, karmaşık mantığa dayalı koşullar oluşturmamıza olanak sağlar.

Ancak unutulmamalıdır ki;

- Çalıştırılırken en verimli yöntem basit düzeydeki mantıksal ifadelerdir.
- Basit düzeydeki mantıksal ifadeleri yazmak ve çalıştırmak daha kolaydır.
- Basit düzeydeki mantıksal ifadeler, çalıştırılırken de en verimli yöntemdir.
- Basit düzeydeki mantıksal ifadelerin değiştirilmesi, düzenlenmesi ve genişletilmesi de daha kolaydır.

Sayfa 141'deki soruları yanıtlayınız.

5.DÖNGÜLER