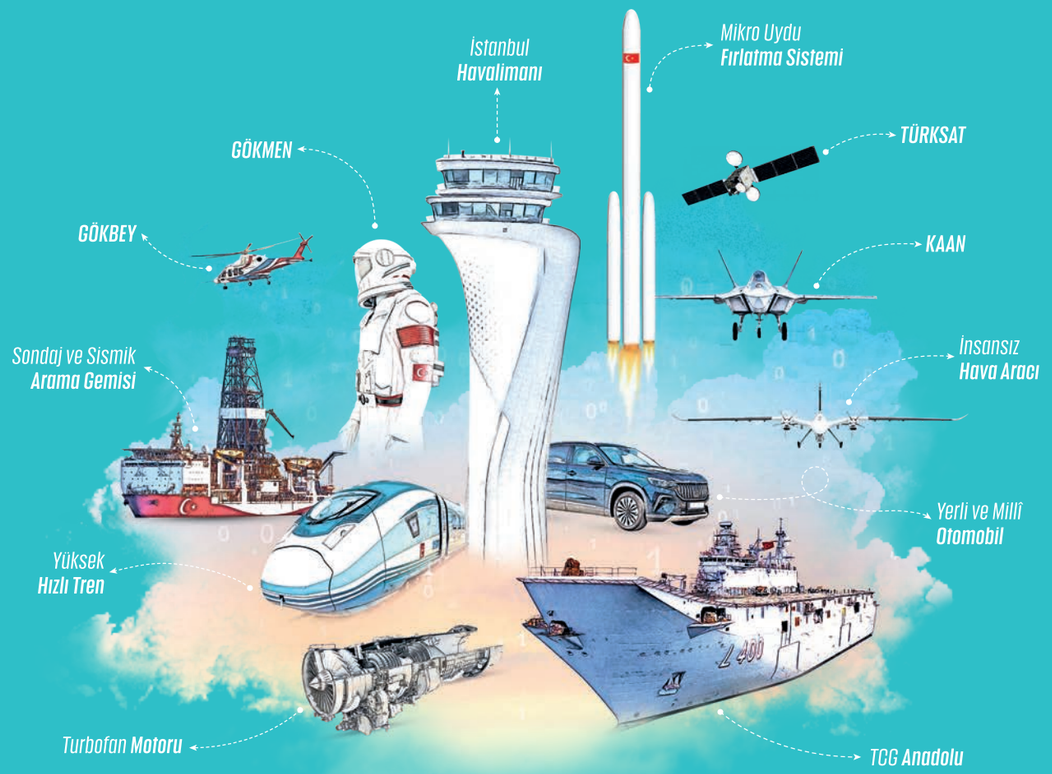
Bu kitaba sığmayan daha neler var!

eba
www.eba.gov.tr

Karekodu okut, bu kitapla ilgili EBA içeriklerine ulaş!

Teknoloji Yolculuğumuz



BU DERS KİTABI MİLLÎ EĞİTİM BAKANLIĞINCA ÜCRETSİZ OLARAK VERİLMİŞTİR.
PARA İLE SATILAMAZ.

ISBN: 978-625-91665-7-5

Bandrol Uygulamasına İlişkin Usul ve Esaslar Hakkında Yönetmeliğin Beşinci Maddesinin İkinci Fıkrası Çerçevesinde Bandrol Taşıması Zorunlu Değildir.

YAPAY ZEKÂ ALANI

ALGORİTMA VE PROGRAMLAMA

9 DERS MATERYALİ

MESLEKİ VE TEKNİK ANADOLU LİSESİ

YAPAY ZEKÂ ALANI

ALGORİTMA VE PROGRAMLAMA



9

Ders Materyali

MESLEKİ VE TEKNİK ANADOLU LİSESİ

YAPAY ZEKÂ ALANI

ALGORİTMA VE PROGRAMLAMA

9

DERS MATERYALİ

YAZARLAR

Abdullah ALAN
Ayhan ÖZREK
Ekrem ORAL

MİLLÎ EĞİTİM BAKANLIĞI YAYINLARI .....: 10840
DERS KİTAPLAR DİZİSİ .....: 3759
Her hakkı saklıdır ve Millî Eğitim Bakanlığına aittir. Ders materyalinin metin, soru ve şekilleri kısmen de olsa hiçbir surette alınıp yayımlanamaz.

HAZIRLAYANLAR

Dil Uzmanı
Tülay SARAYLIOĞLU

Program Geliştirme Uzmanı
Hakan KARADAĞLI

Rehberlik Uzmanı
Zeynep GÖÇ HİSARKAYA

Ölçme ve Değerlendirme Uzmanı
Günay DURUCAN

Görsel Tasarım Uzmanı
Burcu ÖZTÜRK

ISBN: 978-625-91665-7-5

Millî Eğitim Bakanlığının 24.12.2020 gün ve 18433886 sayılı oluru ile Meslekî ve Teknik Eğitim Genel Müdürlüğünce ders materyali olarak hazırlanmıştır.



İSTİKLÂL MARŞI

Korkma, sönmez bu şafaklarda yüzen al sancak;
Sönmeden yurdumun üstünde tüten en son ocak.
O benim milletimin yıldızıdır, parlayacak;
O benimdir, o benim milletimindir ancak.

Çatma, kurban olayım, çehreni ey nazlı hilâl!
Kahraman ırkıma bir gül! Ne bu şiddet, bu celâl?
Sana olmaz dökülen kanlarımız sonra helâl.
Hakkıdır Hakk'a tapan milletimin istiklâl.

Ben ezelden beridir hür yaşadım, hür yaşarım.
Hangi çılgın bana zincir vuracakmış? Şaşarım!
Kükremiş sel gibiyim, bendimi çiğner, aşarım.
Yırtarım dağları, enginlere sığmam, taşarım.

Garbın âfâkını sarmışsa çelik zırhlı duvar,
Benim iman dolu göğsüm gibi serhaddim var.
Ulusun, korkma! Nasıl böyle bir imanı boğar,
Medeniyyet dediğin tek dişi kalmış canavar?

Arkadaş, yurduma alçakları uğratma sakın;
Siper et gövdeni, dursun bu hayâsızca akın.
Doğacaktır sana va'dettiği günler Hakk'ın;
Kim bilir, belki yarın, belki yarından da yakın.

Bastığın yerleri toprak diyerek geçme, tanı:
Düşün altındaki binlerce kefensiz yatanı.
Sen şehit oğlusun, incitme, yazıktır, atanı:
Verme, dünyaları alsan da bu cennet vatanı.

Kim bu cennet vatanın uğruna olmaz ki feda?
Şüheda fışkıracak toprağı sıksan, şüheda!
Cânı, cânânı, bütün varımı alsın da Huda,
Etmesin tek vatanımdan beni dünyada cüda.

Ruhumun senden İlâhî, şudur ancak emeli:
Değmesin mabedimin göğsüne nâmahrem eli.
Bu ezanlar -ki şehadetleri dinin temeli-
Ebedî yurdumun üstünde benim inlemeli.

O zaman vecd ile bin secde eder -varsa- taşım,
Her cerîhamdan İlâhî, boşanıp kanlı yaşım,
Fışkırır ruh-ı mücerret gibi yerden na'şım;
O zaman yükselerek arşa değer belki başım.

Dalgalan sen de şafaklar gibi ey şanlı hilâl!
Olsun artık dökülen kanlarımın hepsi helâl.
Ebediyyen sana yok, ırkıma yok izmihlâl;
Hakkıdır hür yaşamış bayrağımın hürriyyet;
Hakkıdır Hakk'a tapan milletimin istiklâl!

Mehmet Âkif Ersoy

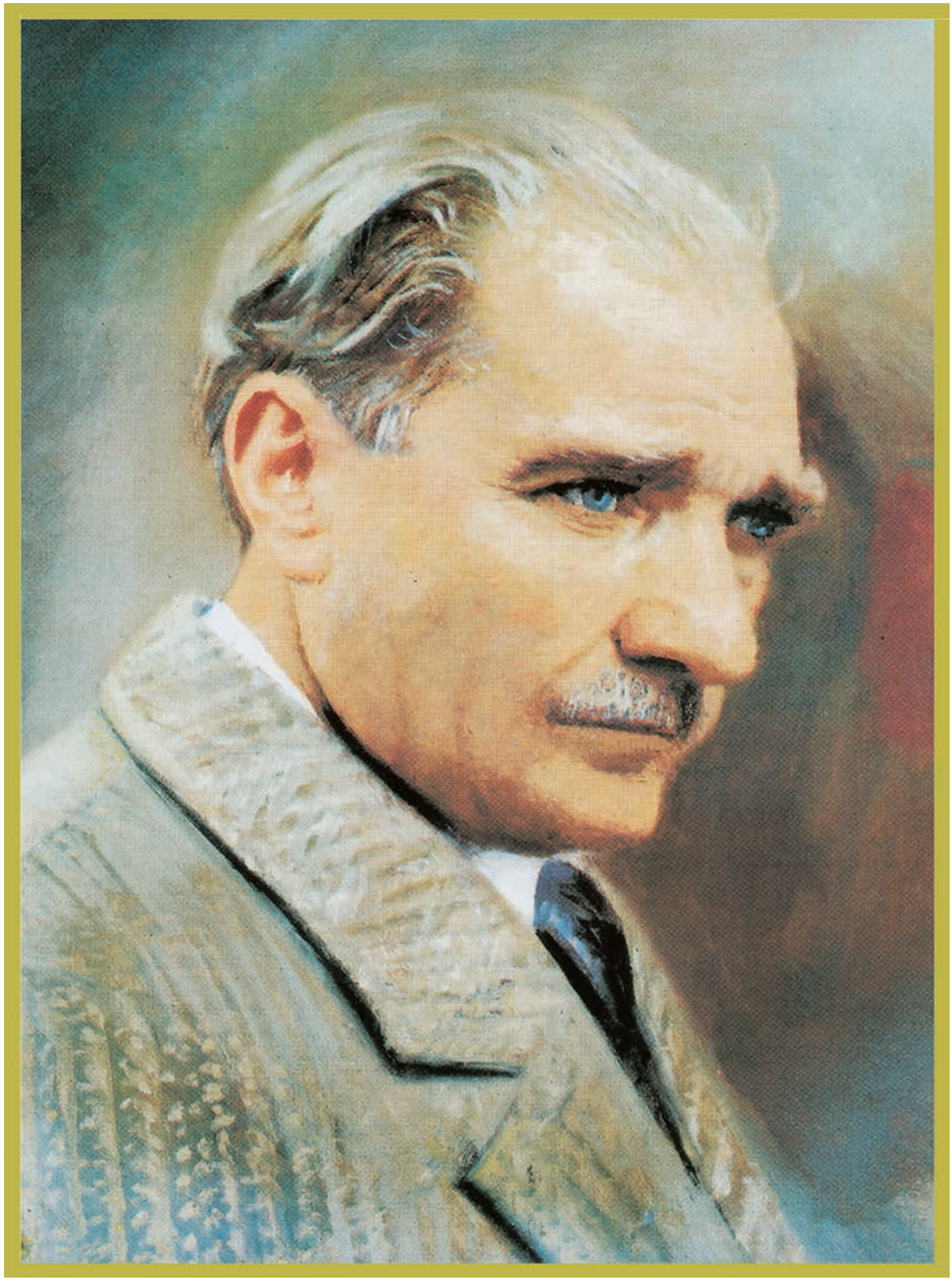
GENÇLİĞE HİTABE

Ey Türk gençliği! Birinci vazifen, Türk istiklâlini, Türk Cumhuriyetini, ilelebet muhafaza ve müdafaa etmektir.

Mevcudiyetinin ve istikbalinin yegâne temeli budur. Bu temel, senin en kıymetli hazinendir. İstikbalde dahi, seni bu hazineden mahrum etmek isteyecek dâhilî ve hâricî bedhahların olacaktır. Bir gün, istiklâl ve cumhuriyeti müdafaa mecburiyetine düşersen, vazifeye atılmak için, içinde bulunacağın vaziyetin imkân ve şeraitini düşünmeyeceksin! Bu imkân ve şerait, çok namüsait bir mahiyette tezahür edebilir. İstiklâl ve cumhuriyetine kastedecek düşmanlar, bütün dünyada emsali görülmemiş bir galibiyetin mümessili olabilirler. Cebren ve hile ile aziz vatanın bütün kaleleri zapt edilmiş, bütün tersanelerine girilmiş, bütün orduları dağıtılmış ve memleketin her köşesi bilfiil işgal edilmiş olabilir. Bütün bu şeraitten daha elîm ve daha vahim olmak üzere, memleketin dâhilinde iktidara sahip olanlar gaflet ve dalâlet ve hattâ hıyanet içinde bulunabilirler. Hattâ bu iktidar sahipleri şahsî menfaatlerini, müstevlîlerin siyasî emelleriyle tevhit edebilirler. Millet, fakr u zaruret içinde harap ve bîtap düşmüş olabilir.

Ey Türk istikbalinin evlâdı! İşte, bu ahval ve şerait içinde dahi vazifen, Türk istiklâl ve cumhuriyetini kurtarmaktır. Muhtaç olduğun kudret, damarlarındaki asil kanda mevcuttur.

Mustafa Kemal Atatürk



MUSTAFA KEMAL ATATÜRK

Ders materyalinin dijital kopyalarında
"İçindekiler" sayfasında başlıklar üzerine tıklayarak
ilgili sayfalara , iç sayfalarda Öğrenme Birimi isimleri
veya numaralarının üzerine tıklayarak "İçindekiler"
sayfasına ulaşabilirsiniz.

İÇİNDEKİLER

DERS MATERYALİNİN TANITIMI....10
1. ÖĞRENME BİRİMİ: İŞ SAĞLIĞI VE GÜVENLİĞİ....12
1.1. İŞ SAĞLIĞI VE GÜVENLİĞİNE İLİŞKİN KAVRAMLAR....14
1.2. ÇALIŞANLAR AÇISINDAN İŞ KAZALARINDAN DOĞABİLECEK HUKUKİ SONUÇLAR....20
1.3. İŞ ORTAMINA İLİŞKİN İŞ SAĞLIĞI VE GÜVENLİĞİ TEDBİRLERİ....23
1.4. İŞ YERİNDE OLUŞABİLECEK İŞ KAZALARINA KARŞI GEREKLİ KİŞİSEL TEDBİRLER....29
1.5. MESLEK HASTALIKLARI VE BU HASTALIKLARDAN KORUNMA....33
ÖLÇME VE DEĞERLENDİRME....40
2. ÖĞRENME BİRİMİ: DİJİTAL PORTFOLYO....44
2.1. DİJİTAL PORTFOLYO PLATFORMU ÇEŞİTLERİ....47
2.2. DİJİTAL PORTFOLYO PLATFORMLARI....50
ÖLÇME VE DEĞERLENDİRME....60
3. ÖĞRENME BİRİMİ: PROGRAMLAMAYA GİRİŞ....64
3.1. BİLGİSAYAR PROGRAMLAMA DİLLERİNİN TEMEL YAPISI....67
3.2. VERİ TİPLERİ....76
3.3. OPERATÖRLERLE MATEMATİKSEL VE MANTIKSAL İŞLEMLER....83
3.4. KOŞULLU İFADELER VE KARAR YAPILARI....89
3.5. DÖNGÜ YAPILARIYLA TEKRARLI İŞLEMLER....96
ÖLÇME VE DEĞERLENDİRME....105
4. ÖĞRENME BİRİMİ: VERİ KOLEKSİYONLARI....110
4.1. LİSTE VERİ YAPILARI....113
4.2. DEMET VERİ YAPISI....129
4.3. KÜME VERİ YAPILARI....136
4.4. SÖZLÜK VERİ YAPISI....144
ÖLÇME VE DEĞERLENDİRME....160
5. ÖĞRENME BİRİMİ: FONKSİYON VE MODÜLER PROGRAMLAMA....164
5.1. FONKSİYONLAR....166
5.2. MODÜLER PROGRAMLAMA....200
ÖLÇME VE DEĞERLENDİRME....207
6. ÖĞRENME BİRİMİ: HATA YAKALAMA İŞLEMLERİ....212
6.1. İSTİSNA İŞLEMLERİ....214
6.2. HATA AYIKLAMA BLOKLARI....234
ÖLÇME VE DEĞERLENDİRME....245
7. ÖĞRENME BİRİMİ: NESNE YÖNELİMLİ PROGRAMLAMA (OOP)....250
7.1. NESNE VE SINIFLAR....253
7.2. MİRAS ALMA İŞLEMLERİ....261
7.3. ÇOK BİÇİMLİ FONKSİYONLAR....266
7.4. SOYUT SINIFLAR....268
ÖLÇME VE DEĞERLENDİRME....276
8. ÖĞRENME BİRİMİ: ARAMA VE SIRALAMA ALGORİTMALARI....280
8.1. ARAMA ALGORİTMALARI....283
8.2. SIRALAMA ALGORİTMALARI....288
ÖLÇME VE DEĞERLENDİRME....301
9. ÖĞRENME BİRİMİ: TEMEL VERİ YAPILARI....306
9.1. VERİ YAPILARI....308
9.2. DOSYA YAPILARI....327
ÖLÇME VE DEĞERLENDİRME....338
KAYNAKÇA....344
GENEL AĞ VE GÖRSEL KAYNAKÇASI KAREKODU....344
CEVAP ANAHTARLARI KAREKODU....344

DERS MATERYALİNİN TANITIMI



Öğrenme biriminin numarasını gösterir.

Öğrenme biriminin adını gösterir.

Öğrenme biriminin karekodunu gösterir.

Öğrenme biriminde yer alan konuları gösterir.

Öğrenme biriminde önemli kavramları gösterir.

Öğrenme biriminde neler öğrenileceğinin ön bilgilerini gösterir.

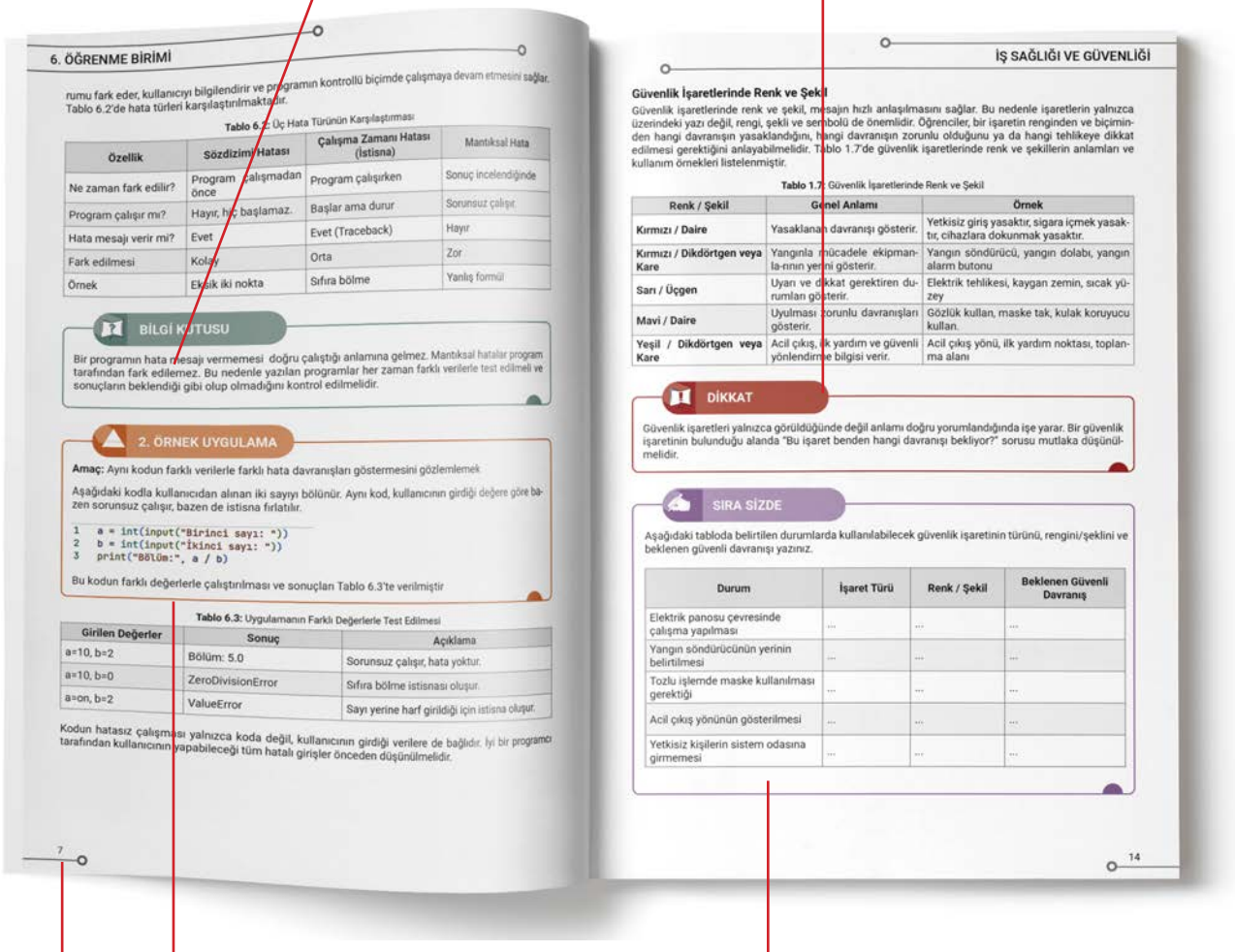
MATERYAL GENELİNDE KULLANILAN İKONLAR VE ANLAMLARI

- Konular
- Hazırlık Çalışmaları
- Dikkat
- Anahtar Kavramlar
- Sıra Sizde
- Örnek Uygulama
- Beklentiler
- Bilgi Kutusu
- Uygulama
- Notlar

* Bu ders materyalinde ölçü birimlerinin uluslararası kısaltmaları kullanılmıştır.

Konu hakkında kritik öneme sahip bilgileri içerir.

Konu hakkında uyulması gereken kuralları tanımlar.



6. ÖĞRENME BİRİMİ

rumu fark eder, kullanıcıyı bilgilendirir ve programın kontrollü biçimde çalışmaya devam etmesini sağlar. Tablo 6.2'de hata türleri karşılaştırılmaktadır.

Tablo 6.2: Üç Hata Türünün Karşılaştırması

Özellik	Sözdizimi Hatası	Çalışma Zamanı Hatası (İstisna)	Mantıksal Hata
Ne zaman fark edilir?	Program çalışmadan önce	Program çalışırken	Sonuç incelendiğinde
Program çalışır mı?	Hayır, hiç başlamaz.	Başlar ama durur	Sorunsuz çalışır.
Hata mesajı verir mi?	Evet	Evet (Traceback)	Hayır
Fark edilmesi	Kolay	Orta	Zor
Örnek	Eksik iki nokta	Sıfıra bölme	Yanlış formül

BİLGİ KUTUSU

Bir programın hata mesajı vermemesi doğru çalıştığı anlamına gelmez. Mantıksal hatalar program tarafından fark edilemez. Bu nedenle yazılan programlar her zaman farklı verilerle test edilmeli ve sonuçların beklendiği gibi olup olmadığını kontrol edilmelidir.

2. ÖRNEK UYGULAMA

Amaç: Aynı kodun farklı verilerle farklı hata davranışları göstermesini gözlemlemek.

Aşağıdaki kodla kullanıcıdan alınan iki sayıyı bölünür. Aynı kod, kullanıcının girdiği değere göre bazen sorunsuz çalışır, bazen de istisna fırlatılır.

```
1  a = int(input("Birinci sayı: "))
2  b = int(input("İkinci sayı: "))
3  print("Bölüm:", a / b)
```

Bu kodun farklı değerlerle çalıştırılması ve sonuçları Tablo 6.3'te verilmiştir

Tablo 6.3: Uygulamanın Farklı Değerlerle Test Edilmesi

Girilen Değerler	Sonuç	Açıklama
a=10, b=2	Bölüm: 5.0	Sorunsuz çalışır, hata yoktur.
a=10, b=0	ZeroDivisionError	Sıfıra bölme istisnası oluşur.
a=on, b=2	ValueError	Sayı yerine harf girildiği için istisna oluşur.

Kodun hatasız çalışması yalnızca koda değil, kullanıcının girdiği verilere de bağlıdır. İyi bir programcı tarafından kullanıcının yapabileceği tüm hatalı girişler önceden düşünülmelidir.

7

İŞ SAĞLIĞI VE GÜVENLİĞİ

Güvenlik İşaretlerinde Renk ve Şekil

Güvenlik işaretlerinde renk ve şekil, mesajın hızlı anlaşılmasını sağlar. Bu nedenle işaretlerin yalnızca üzerindeki yazı değil, rengi, şekli ve sembolü de önemlidir. Öğrenciler, bir işaretin renginden ve biçiminden hangi davranışın yasaklandığını, hangi davranışın zorunlu olduğunu ya da hangi tehlikeye dikkat edilmesi gerektiğini anlayabilmelidir. Tablo 1.7'de güvenlik işaretlerinde renk ve şekillerin anlamları ve kullanım örnekleri listelenmiştir.

Tablo 1.7: Güvenlik İşaretlerinde Renk ve Şekil

Renk / Şekil	Genel Anlamı	Örnek
Kırmızı / Daire	Yasaklanan davranışı gösterir.	Yetkisiz giriş yasaktır, sigara içmek yasaktır, cihazlara dokunmak yasaktır.
Kırmızı / Dikdörtgen veya Kare	Yangınla mücadele ekipmanlarının yerini gösterir.	Yangın söndürücü, yangın dolabı, yangın alarm butonu
Sarı / Üçgen	Uyarı ve dikkat gerektiren durumları gösterir.	Elektrik tehlikesi, kaygan zemin, sıcak yüzey
Mavi / Daire	Uyulması zorunlu davranışları gösterir.	Gözlük kullan, maske tak, kulak koruyucu kullan.
Yeşil / Dikdörtgen veya Kare	Acil çıkış, ilk yardım ve güvenli yönlendirme bilgisi verir.	Acil çıkış yönü, ilk yardım noktası, toplanma alanı

DİKKAT

Güvenlik işaretleri yalnızca görüldüğünde değil anlamı doğru yorumlandığında işe yarar. Bir güvenlik işaretinin bulunduğu alanda "Bu işaret benden hangi davranışı bekliyor?" sorusu mutlaka düşünülmelidir.

SIRA SİZDE

Aşağıdaki tabloda belirtilen durumlarda kullanılabilecek güvenlik işaretinin türünü, rengini/şeklini ve beklenen güvenli davranışı yazınız.

Durum	İşaret Türü	Renk / Şekil	Beklenen Güvenli Davranış
Elektrik panosu çevresinde çalışma yapılması	...	...	...
Yangın söndürücünün yerinin belirtilmesi	...	...	...
Tozlu işlemde maske kullanılması gerektiği	...	...	...
Acil çıkış yönünün gösterilmesi	...	...	...
Yetkisiz kişilerin sistem odasına girmemesi	...	...	...

14

Konuyu pekiştirmek için öğrencinin gerçekleştirdiği uygulamaları gösterir.

Konuyla ilgili bilgi ve becerilerin uygulanabileceği etkinlikleri gösterir.

Sayfa numarasını gösterir.



KONULAR

1.1. İŞ SAĞLIĞI VE GÜVENLİĞİNE İLİŞKİN KAVRAMLAR

1.2. ÇALIŞANLAR AÇISINDAN İŞ KAZALARINDAN DOĞABİLECEK HUKUKİ SONUÇLAR

1.3. İŞ ORTAMINA İLİŞKİN İŞ SAĞLIĞI VE GÜVENLİĞİ TEDBİRLERİ

1.4. İŞ YERİNDE OLUŞABİLECEK İŞ KAZALARINA KARŞI GEREKLİ KİŞİSEL TEDBİRLER

1.5. MESLEK HASTALIKLARI VE BU HASTALIKLARDAN KORUNMA

ANAHTAR KAVRAMLAR

- ergonomi
- güvenlik işaretleri
- iş kazası
- kişisel koruyucu donanım
- meslek hastalığı
- önlem
- ramak kala olay
- risk
- tahliye planı
- tehlike
- tehlikeli davranış
- tehlikeli durum
- yasal düzenleme



1. ÖĞRENME BİRİMİ

İŞ SAĞLIĞI VE GÜVENLİĞİ

BEKLENTİLER

Bu öğrenme biriminde sizden

- iş sağlığı ve güvenliğine ilişkin kavramları açıklamanız
- çalışanlar açısından iş kazalarından doğabilecek hukuki sonuçları açıklamanız
- iş ortamına ilişkin iş sağlığı ve güvenliği tedbirlerini belirlemeniz
- iş yerinde oluşabilecek iş kazalarına karşı gerekli kişisel tedbirleri almanız
- meslek hastalıklarının temel risk etkenlerini ve bu hastalıklardan korunma yollarını açıklamanız

beklenmektedir.

HAZIRLIK ÇALIŞMALARI

1. Aşağıda iş sağlığı ve güvenliği ile ilgili bir metin verilmiştir.

Öğrenciler derse başlamadan önce bilgisayar laboratuvarındaki yerlerine geçerler. Masalardan birinin yanında uzatma kablosu gelişigüzel durmaktadır. Öğrencilerden biri, bu durumu fark etse de önemsemez. Öne eğilerek uzun süre çalışan başka bir öğrenci boyun ve sırt bölgesinde rahatsızlık hissettiği için acıyla kalkıp sandalyesini sınıfın geçiş alanına bırakır. Bu durum arkadaşlarının geçişini zorlaştırır hem de onların kabloya takılma tehlikesine yol açar.

Verilen metne göre aşağıdaki soruları cevaplayınız.

a) Metinde güvenlik açısından uygun olmayan durumlar nelerdir?

b) Bu ortamda öncelikle hangi sorunu düzeltirdiniz?

c) Metindeki sorunlar giderilmezse hangi kazalar veya sağlık sorunları yaşanabilir?

BAŞLARKEN

Günlük yaşamda küçük görünen bazı durumlar, çalışma ortamlarında beklenenden daha büyük sonuçlara yol açabilir. Açıkta bırakılan bir kablo, kapatılan bir acil çıkış yolu, dikkate alınmayan bir uyarı işareti, uygun kullanılmayan bir cihaz ya da uzun süre devam eden yanlış çalışma alışkanlığı güvenli çalışmayı olumsuz etkileyebilir. Atölye, laboratuvar ve iş yerlerinde güvenli bir ortam oluşturmak yalnızca kuralları bilmekle değil, çevredeki değişiklikleri fark etmekle, risk oluşturabilecek durumları zamanında bildirmekle ve doğru davranışı seçmekle mümkündür. Bu öğrenme biriminde çalışma ortamlarında karşılaşılabilecek tehlikeleri fark etmeyi, riskleri değerlendirmeyi, kişisel ve ortamla ilgili tedbirleri ayırt etmeyi ve güvenli çalışma kültürüne katkı sağlayacak davranışlar geliştirilecektir.

1.1. İŞ SAĞLIĞI VE GÜVENLİĞİNE İLİŞKİN KAVRAMLAR

İş sağlığı ve güvenliği (**İSG**), çalışma ortamındaki kişilerin sağlığını korumaya ve güvenliğini sağlamaya yönelik kural, uygulama ve önlemler bütünüdür. Bu kavram yalnızca bir kazadan sonra yapılacakları değil kaza ve zararları önlemek için önceden alınması gereken tedbirleri de kapsar. Bir bilgisayar laboratuvarında güvenli çalışma; masa düzeni, kablo bağlantıları, prizlerin durumu, cihazların uygun yerleşimi, havalandırma, ekran ile oturma düzeni, acil çıkış yolları, güvenlik işaretleri gibi birçok unsuru içerir.

İş sağlığı ve güvenliğinde kullanılan temel kavramların iyi **ayırt edilmesi gerekmektedir**. Tehlike, risk, ramak kala olay ve iş kazası bir olayın kazaya dönüşme sürecini anlamaya yardımcı olurken; meslek hastalığı uzun süreli çalışma koşullarının sağlık üzerindeki etkilerini açıklar. Bu kavramları doğru ayırt etmek, uygun önlemi seçmenin ilk adımıdır. Tablo 1.1'de iş sağlığı ve güvenliğinde temel kavramlar listelenmiştir.

Tablo 1.1: İş Sağlığı ve Güvenliğinde Temel Kavramlar ve Açıklamaları

Kavram	Açıklama	Örnek
Tehlike	Zarar verme potansiyeli bulunan durum, nesne veya koşuldur.	hasarlı priz, açıkta duran kablo, aşırı ısınan cihaz, kaygan zemin
Risk	Tehlikenin zarara yol açma olasılığı ve ortaya çıkarabileceği sonuçtur.	hasarlı priz nedeniyle elektrik çarpması veya yangın çıkması olasılığı
Ramak Kala Olay	Yaralanma, sağlık sorunu veya maddi hasar oluşmadan **atlatılan** ancak kazaya **dönüşme ihtimali bulunan** bir olaydır.	öğrencinin kaygan zeminde düşmeden dengesini toparlaması
İş Kazası	Çalışma sırasında veya işle bağlantılı bir durumda meydana gelen ve kişiye zarar veren olaydır.	cihaz taşırken düşme, elektrik çarpması yaşama
Meslek Hastalığı	Çalışma koşulları veya yapılan işin niteliği nedeniyle zaman içinde ortaya çıkan sağlık sorunudur.	uzun süre uygun olmayan oturuşa bağlı kas-iskelet sistemi zorlanması
Önlem	Tehlike veya riski azaltmak için yapılan düzenlemedir.	kabloların sabitlenmesi, prizlerin kontrol edilmesi, cihazların uygun yere yerleştirilmesi
Güvenli Çalışma Ortamı	Tehlikelerin azaltıldığı, kuralların uygulandığı ve çalışmanın güvenle sürdürüldüğü ortamdır.	geçiş yolları açık, cihazların düzenli, güvenlik işaretlerinin görünür olduğu laboratuvar

Tehlikeli Durum ve Tehlikeli Davranış

İş kazaları yalnızca ortamdan kaynaklanmaz, kişilerin davranışları da kazaya neden olabilir. Bu nedenle iş sağlığı ve güvenliğinde tehlikeli durum ve tehlikeli davranış birlikte ele alınır.

Tehlikeli durum, çalışma ortamında güvenliği bozan koşullardır. Kaygan zemin, yetersiz aydınlatma, arızalı priz, düzensiz kablo yerleşimi, geçiş yolunun kapatılması, yangın ekipmanının önünün engellenmesi veya cihazların uygun olmayan biçimde yerleştirilmesi tehlikeli duruma örnektir.

Tehlikeli davranış, kişinin güvenliğini tehlikeye atan hatalı davranıştır. Güvenlik işaretlerini dikkate almamak, öğretmen izni olmadan cihaz bağlantılarını değiştirmek, gerekli kişisel koruyucu donanımı kullanmamak, çalışan cihaz üzerinde dikkatsiz işlem yapmak veya çalışma ortamında dikkat dağıtıcı davranışlarda bulunmak tehlikeli davranışa örnektir. Tablo 1.2'de tehlikeli durum ve tehlikeli davranış örnekleri listelenmiştir.

Tablo 1.2: Tehlikeli Durum ve Tehlikeli Davranışlar

Kavram	Kaynağı	Örnek
Tehlikeli Durum	Ortam, ekipman veya çalışma düzeninden kaynaklanır.	acil çıkış yolunun önünde kutuların bulunması, bilgisayar kasasının havalandırma bölümünün kapalı kalması ve yangın söndürücünün görünmeyecek şekilde eşya arkasında kalması
Tehlikeli Davranış	Kişinin güvenli olmayan davranışından kaynaklanır.	ıslak elle prize dokunulmaya çalışılması, güvenlik işareti bulunan alanda dikkatsiz davranılması ve öğretmen izni olmadan cihaz bağlantılarının değiştirilmesi

Bazı durumlarda tehlikeli durum ve tehlikeli davranış birlikte ortaya çıkabilir. Örneğin arızalı bir priz tehlikeli durum, bu prizi kullanmaya devam etmek ise tehlikeli davranış olarak değerlendirilebilir. Bu nedenle güvenli çalışma için ortamın düzenli olması ve kişilerin güvenli davranış göstermesi gerekir.

İş Kazalarının Nedenleri

İş kazaları çoğu zaman tek bir nedenle ortaya çıkmaz. Tehlikeli durumlar, tehlikeli davranışlar, eğitim eksikliği, acelecilik, dalgınlık, uygun olmayan ortam koşulları ve ekipman hataları birlikte kazaya zemin hazırlayabilir. Bu nedenle güvenli çalışma yalnızca ortamı düzenlemekten ibaret değildir. Öğrencinin kendi davranışını kontrol etmesi, yönergelere uyması, gördüğü eksiklikleri bildirmesi ve riskleri önemsemesi gibi güvenli davranışları da içerir. Tablo 1.3'te iş kazalarının nedenleri listelenmiştir.

Tablo 1.3: İş Kazalarının Nedenleri

Kaza Nedeni	Açıklama	Örnek
İnsana Bağlı Nedenler	Dikkatsizlik, acelecilik, yorgunluk, kuralları önemsememe veya yeterli bilgiye sahip olmama gibi nedenlerdir.	güvenlik işaretine rağmen cihaz bağlantısınnı izinsiz değiştirilmesi
Çevresel Nedenler	Ortam düzeni, aydınlatma, havalandırma, gürültü, zemin ve geçiş yolları gibi koşullardan kaynaklanır.	kaygan zemin, yetersiz aydınlatma, acil çıkış yolunun kapatılması
Ekipman Kaynaklı Nedenler	Hasarlı, bakımsız, uygun yerleştirilmemiş veya güvenlik açısından kontrol edilmemiş araç gereçten kaynaklanır.	aşırı ısınan cihaz, arızalı veya topraksız priz, gevşek bağlantılı kablo
Planlama ve Denetimden Kaynaklanan Nedenler	Görev paylaşımı, bilgilendirme, bakım, kontrol ve denetim eksikliklerinden kaynaklanabilir.	risk kontrol formunun tutulmaması, cihaz bakımının gecikmesi, güvenlik uyarılarının görünür olmaması

İş kazalarının önlenebilmesi için yalnızca tehlikeyi fark etmek yeterli değildir. Tehlikenin neden oluştuğunu anlamak, benzer durumların tekrar yaşanmasını önlemek açısından önemlidir.

Ramak Kala Olay

Yaralanma, sağlık sorunu veya maddi hasar oluşmadan atlatılan ancak farklı sonuçlanması hâlinde kazaya yol açabilecek olaylara **ramak kala olay** adı verilir. Ramak kala olaylarda zarar meydana gelmemiş olabilir ancak bu durum, ortamda tehlike veya risk bulunmadığı anlamına gelmez.

Bir öğrencinin kaygan zeminde dengesini kaybedip düşmeden tutunması, bir cihaz kablosundan kıvılcım çıkmasına rağmen yangın oluşmaması ya da masa kenarında duran bir cihazın düşmek üzereyken fark edilip güvenli yere alınması ramak kala olaya örnek olarak verilebilir.

Ramak kala olayların öğretmene veya sorumlu kişiye bildirilmesi önemlidir. Çünkü bu olaylar, daha büyük kazalar yaşanmadan önce tehlikeyi fark etme ve önlem alma fırsatı verir. Güvenli çalışma kültüründe yalnızca gerçekleşen kazalar değil kazaya dönüşmeden atlatılan olaylar da dikkate alınır.

BİLGİ KUTUSU

Kaza piramidi yaklaşımı; ciddi kazaların öncesinde daha küçük olayların güvensiz davranışların veya ramak kala durumların görülebileceğini hatırlatır. Bu nedenle "Bir şey olmadı." düşüncesiyle geçiştirilen olaylar da önemsenmeli ve gerekli önlemler alınmalıdır.

İş Kazası ve Meslek Hastalığı

Çalışma sırasında ya da işle bağlantılı bir durumda meydana gelen, kişide bedensel veya ruhsal zarara yol açabilen olaylar iş kazalarıdır. Laboratuvarda cihaz taşırken düşmek, elektrik çarpması yaşamak veya geçiş alanında bulunan bir nesneye takılarak yaralanmak iş kazasına örnek olabilir.

Yapılan işin niteliği veya çalışma koşulları nedeniyle zaman içinde ortaya çıkan sağlık sorunu **meslek hastalığı** olarak nitelendirilir. Uygun olmayan duruşla uzun süre çalışmak, tozlanan bilgisayar kasasını temizlerken koruyucu ekipman kullanmak, yüksek sesli cihazların yanında kulak koruyucu olmadan bulunmak ve ekran karşısında ara vermeden çalışmak zamanla sağlık sorunlarına neden olabilir.

İş kazaları genellikle kısa sürede fark edilirken, meslek hastalıkları çoğu kez zaman içinde gelişir. Bu nedenle iş sağlığı ve güvenliği hem ani kazaları hem de uzun süreli sağlık risklerini önlemeyi amaçlar. Tehlike, risk, ramak kala olay, iş kazası ve meslek hastalığı arasındaki ilişki Görsel 1.1'de örnek bir bilgisayar laboratuvarı üzerinden gösterilmiştir.



Görsel 1.1: Tehlike, risk, ramak kala olay, iş kazası ve meslek hastalığı ilişkisi

SIRA SİZDE

Aşağıdaki durumları inceleyiniz. Her durum için tehlike, risk, ramak kala olay/iş kazası ilişkisini kurunuz ve alınabilecek önlemi yazınız.

Durum	Kavram Zinciri	Önlem
Çalışma ortamında bir kişinin çantasını geçiş yoluna bırakması	Tehlike: Risk: Önlem Alınmazsa:	
Projektör kablosunun masa kenarından sarkması ve kişinin kabloya takılmadan son anda durması	Tehlike: Ramak Kala Olay:	
Yetkili kişi onayı olmadan bilgisayar kasasının iç bağlantılarına müdahale edilmesi	Tehlike: Risk: İş Kazası İhtimali:	
Sıvı dolu bir şişenin bilgisayar ve prizlerin bulunduğu masanın üzerine bırakılması	Tehlike: Risk: Acil Durumda Sonuç:	

Güvenlik Kültürü

Güvenlik kültürü, sağlıklı ve güvenli davranışların alışkanlık hâline gelmesidir. Güvenlik kültürü gelişmiş bir ortamda kişiler yalnızca kendi güvenliklerini değil birlikte çalıştıkları kişilerin güvenliğini de düşünür.

Olumlu güvenlik kültüründe tehlikeler gizlenmez, ramak kala olaylar küçümsenmez, güvenlik işaretleri dikkate alınır ve gerekli durumlar öğretmene ya da sorumlu kişiye bildirilir. Olumsuz güvenlik kültüründe ise "Bana bir şey olmaz.", "Hep böyle yapıyoruz." veya "Zaman kaybetmeyelim." gibi düşünceler riskleri artırabilir. Tablo 1.4'te güvenlik kültürüyle ilgili özellik ve davranışlar listelenmiştir.

Tablo 1.4: Güvenlik Kültürü İle İlgili Özellik ve Davranışlar

Güvenlik Kültürü Boyutu	Temel Özellik	Örnek Davranış
Olumlu Güvenlik Kültürü	Riskleri fark eden, bildiren ve önlem alan anlayıştır.	Hasarlı prizi kullanılmadan öğretmene haber verilmesi
Olumsuz Güvenlik Kültürü	Riskleri önemsemeyen veya görmezden gelen anlayıştır.	Koruyucu donanımı rahatsız ediyor diye kullanmamak
Ortak Sorumluluk	Güvenliğin yalnızca bir kişiye bırakılmadığı anlayıştır.	Grup çalışmasında acil çıkış yolunun açık tutulması

Güvenlik kültürü, kuralların yalnızca bilinmesiyle değil günlük uygulamalarda düzenli olarak yerine getirilmesiyle gelişir. İş sağlığı ve güvenliği kavramları birbirinden bağımsız değildir. Çalışma ortamında bir tehlike fark edildiğinde bu tehlikenin oluşturabileceği risk değerlendirilir. Ardından uygun önlemler alınarak güvenli çalışma ortamı desteklenir. Önlem alınmadığında ise ramak kala olay, iş kazası, uzun süreli maruziyet sonucunda meslek hastalığı gibi olumsuz sonuçlar ortaya çıkabilir (Görsel 1.2).



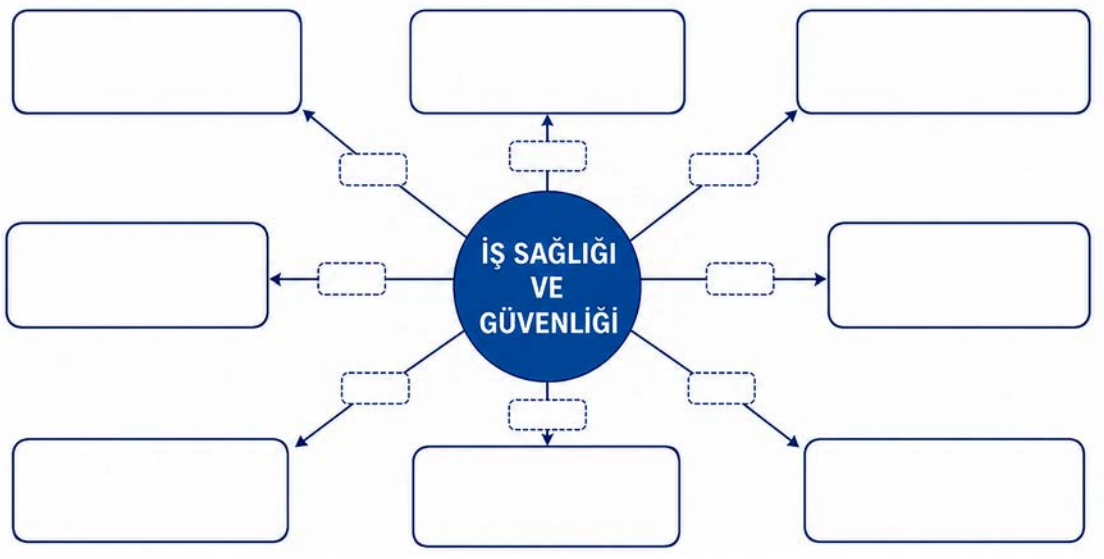
Görsel 1.2: Ramak kala olaylarının bildirilmesi

SIRA SİZDE

İş sağlığı ve güvenliği kavramları için hazırlanan yarı yapılandırılmış kavram haritasını inceleyiniz. Verilen kavramları uygun kutulara yerleştiriniz. Boş bırakılan yerlere kavramlar arasındaki ilişki ifadelerini yazınız.

Kullanılacak Kavramlar: tehlike, risk, ramak kala olay, iş kazası, meslek hastalığı, önlem, güvenli çalışma ortamı, kişisel koruyucu donanım, acil durum önlemleri

Kullanılabilecek İlişki İfadeleri: neden olabilir, artırabilir, azaltır, önlemeye yardımcı olur, sonucunda ortaya çıkabilir, bildirilmelidir, korunmayı destekler



BİLGİ KUTUSU

İş sağlığı ve güvenliği kavramlarını bilmek, çalışma ortamında doğru davranışı seçmeye yardımcı olur. Bilişim laboratuvarlarında donanım montajı, cihaz temizliği, ağ bağlantısı, güç bağlantısı ve ekipman kullanımı gibi işlemler yetkili kişi yönergesine göre yapılmalıdır. Enerji bağlantısı kesilmeden kasa içi müdahale yapılmamalı, hassas elektronik parçalarla çalışırken antistatik önlem alınmalı, ağ cihazlarının bağlantıları izinsiz değiştirilmemeli ve kablolar düzenli biçimde etiketlenmelidir. Arıza, yanık kokusu, duman, aşırı ısınma, güvenlik işareti eksikliği veya yetkisiz müdahale fark edildiğinde durum sorumlu kişiye bildirilmelidir.

1. ETKİNLİK: İSG Kavramlarını Dijital Haritada Yapılandırıyorum

Süre: 40 dakika

Amaç: İş sağlığı ve güvenliğiyle ilgili kavramları ana ve alt kavram ilişkilerine göre çözümlemek

Yönerge: Yazılan adımları takip ederek etkinliği tamamlayınız.

1. Adım: Sıra Sizde bölümünde tamamladığınız kavram haritasından yararlanınız. Öğretmeninizin belirlediği dijital kavram haritası aracını veya sunu yazılımını kullanarak iş sağlığı ve güvenliği kavramlarını hiyerarşik biçimde düzenleyiniz.

Dijital kavram haritanızda aşağıdaki unsurlara yer veriniz.

- Merkez kavram olarak iş sağlığı ve güvenliği
- Ana ve alt kavramlar arasındaki hiyerarşik ilişki
- Tehlike, risk, ramak kala olay, iş kazası ve meslek hastalığı kavramları
- Önlem, kişisel koruyucu donanım ve güvenli çalışma ortamı kavramları
- Kavramlar arasındaki bağlantıları açıklayan kısa ilişki ifadeleri

2. Adım: Haritanızda ana kavramı, alt kavramları ve kavramlar arasındaki ilişki ifadelerini açıkça gösteriniz.

3. Adım: Hazırladığınız çalışmayı bir arkadaşınızın çalışmasıyla karşılaştırınız ve gerekli gördüğünüz düzenlemeleri yapınız.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Kontrol Listesi** ile değerlendirilecektir. Çalışmanızı hazırlarken değerlendirme ölçütlerini dikkate alınız. Verilen karekodu kullanarak kontrol listesine ulaşabilirsiniz.



Kontrol listesine ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



1.2. ÇALIŞANLAR AÇISINDAN İŞ KAZALARINDAN DOĞABİLECEK HUKUKİ SONUÇLAR

İş kazaları yalnızca çalışanın sağlığını etkileyen olaylar değildir. Bir iş kazası yaşandığında olayın neden gerçekleştiği, gerekli güvenlik önlemlerinin alınıp alınmadığı ve ilgili kişilerin sorumluluklarını yerine getirip getirmediği de değerlendirilir. Bu değerlendirme sonucunda bildirim, sağlık hizmeti, sigorta işlemleri, iş göremezlik, tazminat, idari yaptırım veya cezai sorumluluk gibi hukuki süreçler gündeme gelebilir. Bu nedenle iş sağlığı ve güvenliği kurallarına uymak hem bireysel güvenlik hem de hukuki sorumluluklar açısından önemlidir.

Yasal Düzenlemelerin Amacı

İş sağlığı ve güvenliğiyle ilgili yasal düzenlemeler, çalışanların sağlığını korumak ve güvenli çalışma ortamı oluşturmak için hazırlanır. Bu düzenlemeler işverenin, çalışanın ve kurumların sorumluluklarını belirler.

Yasal düzenlemelerin temel amacı, kazalar meydana geldikten sonra yalnızca sonuçlarla ilgilenmek değildir. Asıl amaç, kazaları ve meslek hastalıklarını ortaya çıkmadan önce önlemeye çalışmaktır.

Bir iş kazası yaşandığında olayın nasıl gerçekleştiği, gerekli önlemlerin alınıp alınmadığı ve ilgili kişilerin sorumluluklarını yerine getirip getirmediği değerlendirilir. İşverenin güvenli çalışma ortamı sağlama, çalışanları bilgilendirme ve gerekli önlemleri alma sorumluluğu vardır. Çalışanın ise verilen eğitimlere uygun davranma, güvenlik kurallarına uyma ve tehlikeli durumları bildirme sorumluluğu bulunur.

SIRA SİZDE

Aşağıdaki tabloya verilen duruma yönelik kaza öncesi sorumlu davranışı ve önlem alınmadığında doğabilecek sonucu yazınız.

Durum	Kaza Öncesi Yapılması Gereken Sorumlu Davranış	Önlem Alınmadığında Doğabilecek Sonuç
1. Bilgisayar laboratuvarında hasarlı priz fark edilmesi		
2. Atölyede acil çıkış kapısının önünde kutular bulunması		
3. Bilgisayar kasası bakımı sırasında yoğun toz oluşması		
4. Laboratuvardaki güvenlik işaretinin yıpranmış veya okunamaz olması		

Türkiye'de İş Sağlığı ve Güvenliğinin Yasal Gelişimi

Türkiye'de iş sağlığı ve güvenliğine ilişkin düzenlemeler zaman içinde gelişmiştir (Görsel 1.3). Bu gelişim, çalışanların korunmasına yönelik yaklaşımın daha sistemli hâle gelmesine katkı sağlamıştır.



Görsel 1.3: Türkiye'de iş sağlığı ve güvenliğine ilişkin önemli yasal gelişmeler

2. ETKİNLİK: Yasal Gelişim Sürecini Zaman Çizelgesine Dönüştürüyorum

Süre: 20 dakika

Amaç: Türkiye'de iş sağlığı ve güvenliğiyle ilgili yasal gelişmeleri tarih sırasına göre ilişkilendirmek ve bu gelişmelerin çalışan güvenliği açısından önemini açıklamak

Yönerge: Türkiye'de iş sağlığı ve güvenliğiyle ilgili Görsel 1.3'te belirtilen üç önemli kanunu araştırınız. Ulaştığınız bilgileri aşağıdaki tabloda tarih sırasına göre düzenleyiniz. Her gelişme için kısa ve anlaşılır bir açıklama yazınız. Hazırladığınız bilgileri dijital ortamda zaman çizelgesi hâline getiriniz. Zaman çizelgenizde tarih, yasal gelişme, kısa açıklama ve çalışan güvenliği açısından önemli bölümlerine yer veriniz.

Tarih	Yasal Gelişme	Kısa Açıklama	Çalışan Güvenliği Açısından Önemi

Değerlendirme: Çalışmanız öğretmeniniz tarafından **Kontrol Listesi** ile değerlendirilecektir. Çalışmanızı planlarken kontrol listesinde yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak kontrol listesine ulaşabilirsiniz.

Kontrol listesine ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

İhmalin Doğurabileceği Sonuçlar

İş sağlığı ve güvenliğinde ihmal, bilinen veya fark edilen bir tehlikeye rağmen gerekli önlemin alınmamasıdır. Yangın söndürücünün önüne eşya koymak, hasarlı kabloyu kullanmaya devam etmek, acil çıkış kapısını kilitli bırakmak veya güvenlik işaretlerini görmezden gelmek ihmale örnek gösterilebilir.

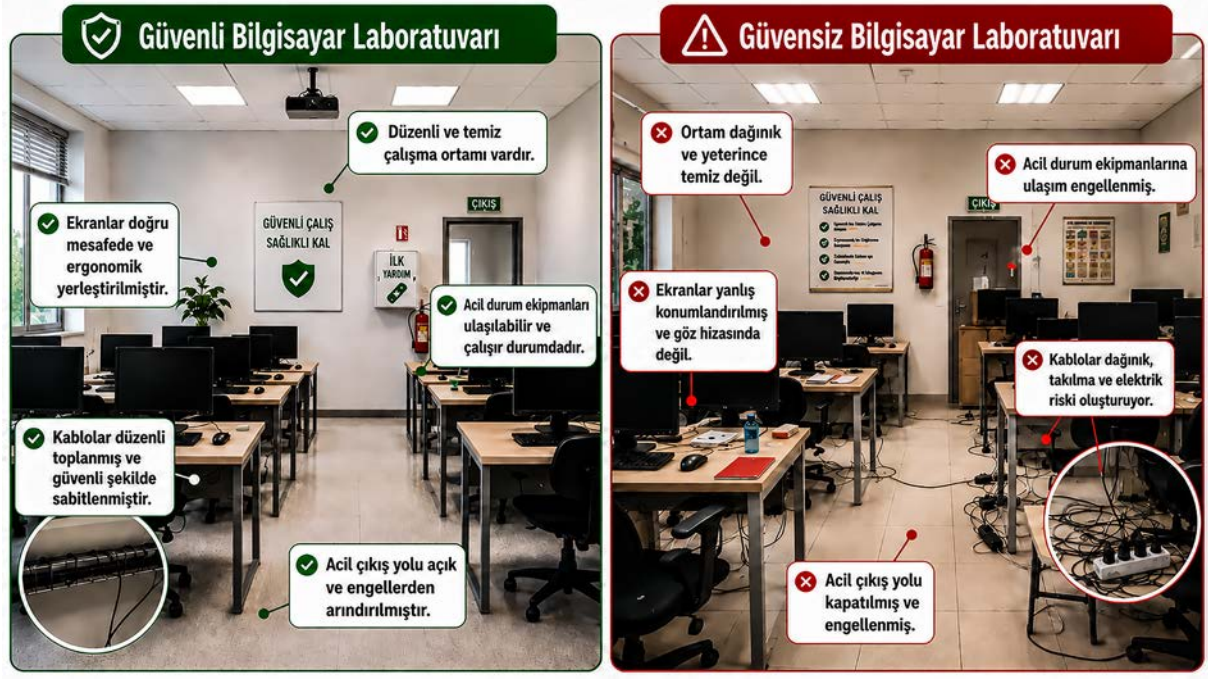
İhmal sonucunda yalnızca bireysel yaralanmalar değil cihaz hasarı, eğitim sürecinin aksaması, maddi kayıp, hukuki sorumluluk ve çalışma ortamında güven duygusunun zayıflaması gibi sonuçlar da ortaya çıkabilir. Bu nedenle güvenli çalışma ortamlarında tehlikeler fark edildiğinde gecikmeden bildirim yapılmalı ve gerekli önlemler alınmalıdır.

1.3. İŞ ORTAMINA İLİŞKİN İŞ SAĞLIĞI VE GÜVENLİĞİ TEDBİRLERİ

Güvenli çalışma alanı; çalışanların veya çalışma ortamındaki kişilerin rahatça hareket edebildiği, araç ve gereçleri doğru biçimde kullanabildiği ve acil durumlarda güvenli şekilde çıkış yapabildiği ortamdır. Laboratuvar ve atölyelerde masa düzeni, cihaz yerleşimi, kablo bağlantıları, havalandırma, aydınlatma, geçiş yolları ve acil çıkış alanları düzenli olmalıdır.

Bilgisayar kasalarının havalandırma boşluklarının kapatılmaması, prizlerin aşırı yüklenmemesi, sıvıların elektronik cihazlardan uzak tutulması ve geçiş yollarının açık bırakılması güvenli çalışma ortamının temel unsurlarıdır. Güvenli çalışma alanı oluşturmak yalnızca sınıfı düzenlemek değil olası tehlikeleri önceden fark ederek uygun tedbiri almaktır.

Güvenli ve güvenli olmayan laboratuvar düzeni arasındaki farkları Görsel 1.4 üzerinden inceleyiniz.



Görsel 1.4: Güvenli ve güvenli olmayan bilgisayar laboratuvarı düzeni

BİLGİ KUTUSU

Güvenli çalışma alanı yalnızca eşyaların düzenli durmasıyla sınırlı değildir. Aydınlatma, havalandırma, sıcaklık, hava akımı, gürültü, temizlik, düzen gibi ortam koşulları da güvenliği ve sağlığı etkiler.

Tablo 1.5'te ortam koşullarının önemi ve alınabilecek önlemler listelenmiştir.

Tablo 1.5: Ortam Koşulları ve Önemi

Ortam Koşulu	Önemi	Alınacak Önlem
Aydınlatma	Yetersiz ışık, dikkati dağıtır ve görmeyi olumsuz etkiler.	Ekran yansıması ve loş alanlar azaltılmalıdır.
Havalandırma	Toz, koku ve hava kalitesinin düşmesinin önlenmesine yardımcı olur.	Bilgisayar kasası temizliği gibi işlemlerde ortam havalandırılmalıdır.
Gürültü	Dikkati dağıtabilir ve uzun süreli maruziyet işitme sağlığını olumsuz etkileyebilir.	Yüksek sesli cihazların yanında gerekli durumlarda kulak koruyucu kullanılmalıdır.
Sıcaklık ve Hava Akımı	Aşırı sıcak, soğuk veya yoğun hava akımı dikkati dağıtarak çalışma verimini olumsuz etkileyebilir.	Ortam sıcaklığı ve hava akımı çalışmaya uygun olmalıdır.
Temizlik ve Düzen	Takılma, kayma, yangın ve hijyen risklerini azaltır.	Koridor, masa altı, kablo geçişleri ve ortak alanlar düzenli tutulmalıdır.

SIRA SİZDE

Bir bilişim laboratuvarında aşağıdaki durumlar gözlemlenmiştir. Gözlemlenen durumların öncelik sırasını aşağıdaki risk önceliklendirme tablosuna yazınız ve kısa gerekçelerini belirtiniz.

Gözlenen Durum	Öncelik Sırası	Gerekçe
1. Elektrik prizinden kıvılcım çıkması		
2. Yangın söndürücünün önünün malzemeyle kapatılması		
3. Bilgisayar kasalarının havalandırma boşluklarının kapanması		
4. Temizlik kimyasalının etiketsiz kapta tutulması		
5. Ekran yüksekliği ve sandalye ayarının uygun olmaması		

Elektrik, Yangın ve Kimyasal Risklere Karşı Tedbirler

Laboratuvarlarda elektrikle çalışan birçok cihaz bulunur. Bu nedenle elektrik bağlantılarının güvenli olması önemlidir. Hasarlı prizler, ezilmiş kablolar, aşırı yüklenmiş çoklu prizler ve sıvı temasına açık elektrikli cihazlar risk oluşturabilir.

Yangın riskine karşı yanıcı maddeler cihazlardan uzak tutulmalı, yangın söndürücülere erişim engellenmemeli ve acil çıkış yolları açık bırakılmalıdır. Kimyasal temizlik maddeleri bilgisayar donanımlarının yakınında açık biçimde bulundurulmamalı ve öğretmen ya da yetkili kişi bilgisi dışında kullanılmamalıdır.

Bilişim atölyelerinde kullanılan toner, sprey temizleyici, basınçlı hava spreyi / tüpü, yüzey temizleyici gibi malzemeler dikkatli kullanılmalıdır. Bu malzemelerin etiketsiz kaplarda saklanması, cihazların üzerine bırakılması veya kapalı ortamda dikkatsiz kullanılması; solunum yolu sorunlarına,cilt temasından kaynaklı risklere ve yangın riskine yol açabilir.

Tablo 1.6'da tehlike kaynakları ve tehlike durumunda alınacak tedbirler listelenmiştir.

Tablo 1.6: Tehlike Kaynakları ve Alınacak Tedbirler

Tehlike Kaynağı	Olası Risk	Alınabilecek Tedbir
Aşırı yüklenmiş çoklu priz	Isınma, elektrik arızası, yangın	Çoklu prizin aşırı yüklenmemesi, zincirleme bağlantı yapılmaması ve durumun öğretmene ya da yetkili kişiye bildirilmesi
Cihaz havalandırmasının kapanması	Aşırı ısınma ve cihaz arızası	Cihaz çevresinde yeterli boşluğun bırakılması ve havalandırma deliklerinin kapatılmaması
Temizlik kimyasalının açıkta bulunması	Cilt teması, solunum rahatsızlığı	Malzemeleri etiketli kaplarda saklanması ve öğretmenin yönergesine uygun kullanılması
Yangın söndürücüye erişimin engellenmesi	Acil durumda müdahalenin gecikmesi	Yangın ekipmanının önünün açık bırakılması
Acil çıkış yönlendirmesinin görünmemesi	Tahliye sırasında karışıklık	Acil çıkış ve yönlendirme işaretlerinin görünürlüğünü sağlamak

Güvenlik İşaretleri ve Anlamları

Güvenlik işaretleri, tehlikeyi önceden fark etmeye ve doğru davranışı seçmeye yardımcı olur. Bu işaretler; kişileri tehlikelere karşı uyaran, yapılması veya yapılmaması gereken davranışları gösteren ve acil durumlarda yönlendirme sağlayan araçlardır. İşaretlerin biçimi, rengi ve üzerinde yer alan sembol, verilmek istenen mesajın hızlı anlaşılmasını kolaylaştırır.

Sağlık ve güvenlik işaretleri; levha, renk, sesli ya da ışıklı sinyal, sözlü uyarı ve el-kol hareketiyle bilgi ile talimat verebilir.

Atölye ve laboratuvarlarda güvenlik işaretleri; yasak davranışları, dikkat edilmesi gereken tehlikeleri, yapılması zorunlu koruyucu önlemleri, acil çıkış yollarını ve yangınla mücadele ekipmanlarının yerini gösterir. Temel işaret türleri Görsel 1.5'te gösterilmiştir.



Görsel 1.5: Güvenlik işaretlerinin renk, şekil ve anlam özellikleri

DİKKAT

Güvenlik işaretleri yalnızca görüldüğünde değil anlamı doğru yorumlandığında işe yarar. Bir güvenlik işaretinin bulunduğu alanda "Bu işaret benden hangi davranışı bekliyor?" sorusu mutlaka düşünülmelidir.

SIRA SİZDE

Aşağıdaki tabloda belirtilen durumlarda kullanılabilecek güvenlik işaretinin türünü, rengini / şeklini ve beklenen güvenli davranışı yazınız.

Durum	İşaret Türü	Renk / Şekil	Beklenen Güvenli Davranış
1. Elektrik panosu çevresinde çalışma yapılması			
2. Yangın söndürücünün yerinin belirtilmesi			
3. Tozlu işlemde maske kullanılması gerektiği			
4. Acil çıkış yönünün gösterilmesi			
5. Yetkisiz kişilerin sistem odasına girmemesi			

BİLGİ KUTUSU

Yasal düzenlemeler, güvenli çalışma ortamının yalnızca kurum tarafından değil, çalışma ortamındaki herkesin sorumlu davranışıyla korunabileceğini gösterir. Bilişim laboratuvarlarında bazı davranışlar küçük görünse de cihaz güvenliği, veri güvenliği ve kişisel güvenlik açısından risk oluşturabilir.

Durum	Sorumlu Davranış	Kaçınılması Gereken Davranış
Bilgisayar kasasına RAM, SSD veya ekran kartı takılması	İşlem yalnızca öğretmen ya da yetkili kişi gözetiminde yapılmalıdır.	Yetki almadan kasa içindeki parçalara dokunmak
Hassas elektronik parçalarla çalışma	Antistatik bileklik veya antistatik mat kullanılmalıdır.	Statik elektrik önlemi almadan donanıma müdahale etmek
Ağ kabini, switch veya modem bağlantılarının kontrolü	Bağlantı değişikliği gerekiyorsa sorumlu kişiden izin alınmalıdır.	Ağ kablosunu rastgele çıkarmak, portları değiştirmek
UPS cihazından uyarı sesi gelmesi	Durum öğretmene veya teknik sorumluya bildirilmelidir.	UPS üzerindeki düğmelere gelişigüzel basmak
Laboratuvarda bakım-onarım formu doldurulması	Arıza, eksik malzeme veya uygunsuz durum kayıt altına alınmalıdır.	Sorunu fark edip bildirmeden çalışmaya devam etmek

İşaretlerin Laboratuvar Ortamında Kullanımı

Güvenlik işaretleri doğru yerde, görünür biçimde ve amacına uygun şekilde kullanılmalıdır. İşaretlerin önünün eşya ile kapanması, yıpranması veya okunamaz hâle gelmesi, acil durumda doğru yönlendirmenin fark edilmesini zorlaştırır.

Bilgisayar laboratuvarı, atölye ve iş yerlerinde acil çıkış, ilk yardım noktası, yangın söndürücü, elektrik tehlikesi ve yetkisiz giriş işaretleri güvenli davranışı destekler ancak işaretlerin bulunması tek başına yeterli değildir. Öğrencilerin bu işaretleri tanıması, anlamını doğru yorumlaması ve gerektiğinde doğru davranışı seçmesi gerekir. Tablo 1.7'de laboratuvar ortamında kullanılacak işaretlerin özellik ve açıklamaları listelenmiştir.

Tablo 1.7: Laboratuvar Ortamında Kullanılacak İşaretlerin Özellik ve Açıklamaları

Dikkat Edilecek Özellik	Açıklama
Anlaşılır Sembol	İşaretin üzerindeki resimli sembol (piktogram) kolay anlaşılır olmalıdır.
Görünürlük	İşaret, görüş seviyesine uygun yükseklikte ve yeterli aydınlatmaya sahip bir noktada bulunmalıdır.
Doğru Konum	İşaret, belirli bir tehlikenin yakınında veya genel tehlike alanının girişinde yer almalıdır.
Dayanıklılık	İşaret, kullanıldığı ortama uygun olmalı darbe ve çevre koşullarına dayanıklı biçimde üretilmelidir.
Erişilebilirlik	İşaretler; özellikle acil çıkış, ilk yardım ve yangın ekipmanı işaretleri kolayca fark edilebilmelidir.

Bilişim Laboratuvarında Etiketleme ve Yetkili Kullanım

Bilişim laboratuvarlarında güvenli çalışma yalnızca uyarı levhalarıyla sağlanmaz. Cihaz, bağlantı noktası ve ekipmanların doğru etiketlenmesi de güvenli kullanımın bir parçasıdır. Ağ kabinindeki switch portlarının, güç bağlantılarının, bakımda olan cihazların ve arızalı ekipmanların açık biçimde işaretlenmesi yanlış müdahaleleri azaltır. Örneğin "Bakımda", "Kullanmayınız", "Yetkili kişi dışında müdahale etmeyiniz", "Enerji bağlantısını kesmeden açmayınız" gibi kısa uyarılar; cihazların yanlış kullanılmasını, bağlantıların karıştırılmasını ve donanım hasarlarını önlemeye yardımcı olur. Etiketi olmayan, bakım durumu bilinmeyen veya kime ait olduğu anlaşılmayan cihazlara müdahale edilmemeli; durum sorumlu kişiye bildirilmelidir.

Güvenli Ekipman Kullanımı

Güvenli ekipman kullanımı yalnızca cihazı çalıştırmak değildir. Cihazı kullanmadan önce, kullanım sırasında ve kullanım sonrasında dikkatli davranmayı gerektirir. Çalışma ortamındaki kişi kullandığı cihazın genel durumunu kontrol etmeli, yetkili kişi onayı olmadan cihaz bağlantılarını değiştirmemeli ve arıza fark ettiğinde durumu sorumlu kişiye bildirmelidir.

Güvenli ekipman kullanımı için şu noktalara dikkat edilmelidir:

- Cihazı kullanmadan önce genel durumunu kontrol etmek
- Cihaz çevresinde sıvı veya gereksiz eşya bulundurmamak
- Cihazın havalandırma alanlarını kapatmamak
- Bilgisayar kasası temizliği, yazıcı bakımı veya yüksek sesli cihaz kullanımı gibi işlemlerde uygun koruyucu donanım kullanmak
- Kullanım sonunda cihazı ve çalışma alanını düzenli bırakmak
- Güvenlik işaretlerini dikkate almak

3. ETKİNLİK: Güvenlik Denetimi Yapıyorum

Süre: 20 dakika

Amaç: Çalışma ortamındaki tehlike ve ramak kala olay ihtimallerini belirleyerek bunlara uygun tedbir ve bildirim yollarını önermek

Yönerge:

1. Adım: Gruplara ayrılınız ve grup içinde görev dağılımı yapınız.

2. Adım: Çalışma alanınızı elektrik bağlantıları, cihaz yerleşimi, havalandırma, geçiş yolları, acil çıkış, güvenlik işaretleri, temizlik, düzen ve ergonomi açısından gözlemleyiniz.

3. Adım: Belirlediğiniz tehlike veya ramak kala olay ihtimallerini aşağıdaki risk kontrol formuna yazınız. Her bulgu için uygulanabilir tedbir, gerekli güvenlik işareti veya sorumluya bildirim yolunu belirtiniz.

4. Adım: Grup olarak en önemli gördüğünüz bir güvenlik sorununu ve çözüm önerinizi sınıfla paylaşınız.

Gözlem Alanı	Belirlenen Tehlike / Ramak Kala İhtimali	Olası Risk	Alınabilecek Tedbir	Gerekli İşaret / Bildirim
Elektrik bağlantıları				
Cihaz yerleşimi ve havalandırma				
Temizlik, kimyasal ve toz oluşumu				
Geçiş yolları ve acil çıkış				
Gürültü, aydınlatma ve ergonomi				
Güvenlik işaretleri ve yangın ekipmanı				

Değerlendirme: Çalışmanız öğretmeniniz tarafından **Kontrol Listesi** ile değerlendirilecektir. Çalışmanızı planlarken kontrol listesinde yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak kontrol listesine ulaşabilirsiniz.



Kontrol listesine ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

1.4. İŞ YERİNDE OLUŞABİLECEK İŞ KAZALARINA KARŞI GEREKLİ KİŞİSEL TEDBİRLER

Güvenli çalışma ortamı yalnızca ortam düzeniyle sağlanmaz. Kişinin kendi davranışını düzenlemesi ve güvenlik kurallarına uygun hareket etmesi de önemlidir. Aceleci davranmak, öğretmen izni olmadan cihaz bağlantılarını değiştirmek, güvenlik işaretlerini dikkate almamak veya çalışma alanını düzensiz bırakmak kişisel sorumluluğun ihmal edilmesine örnektir.

Güvenli davranış; durumu fark etmeyi, olası sonuçları düşünmeyi, doğru karar vermeyi ve uygun şekilde hareket etmeyi gerektirir. Kişi, karşılaştığı durumu değerlendirirken "Bu davranış bana, arkadaşlarıma veya çalışma ortamına zarar verebilir mi?" sorusunu kendine sormalıdır.

Kişisel Koruyucu Donanımı Doğru Seçme

Kişisel koruyucu donanımlar, karşılaşılan tehlike türüne göre seçilmelidir. Bazı temel donanım örnekleri Görsel 1.6'da verilmiştir.



Görsel 1.6: Kişisel koruyucu donanım örnekleri

Kişisel koruyucu donanım, kişiyi belirli tehlikelere karşı korumaya yardımcı olan araç gereçlerin genel adıdır ancak her donanım her tehlike için uygun değildir. Örneğin tozlu ortamlarda maske, yüksek sesli cihaz kullanımında kulak koruyucu, göze parça sıçrama riski bulunan durumlarda ise koruyucu gözlük kullanılmalıdır.

Güvenli çalışma anlayışında öncelik, tehlikeyi kaynağında azaltmak ve ortamı güvenli hâle getirmektir. Havalandırma sağlamak, cihazları güvenli şekilde yerleştirmek, kabloları düzenlemek, yangın ekipmanlarını erişilebilir tutmak ve gerekli uyarı sistemlerini kullanmak bu kapsamdaki önlemlere örnek verilebilir. Kişisel koruyucu donanım ise bu önlemleri tamamlayan kişisel korunma basamağıdır.

Kişisel Güvenlik İçin Dikkat Edilecek Noktalar

Kişisel tedbirler; öğrencinin çalışma sırasında uygun kıyafet seçmesi, dikkatli davranması, çalışma alanını düzenli kullanması ve riskli durumları zamanında bildirmesi de güvenli çalışma davranışının bir parçasıdır.

Tablo 1.8'de kişisel güvenlik için dikkat edilecek noktalar ve güvenli davranışlar listelenmiştir.

Tablo 1.8: Kişisel Güvenlik İçin Dikkat Edilecek Noktalar

Dikkat Edilecek Nokta	Güvenli Davranış
Ramak kala olay ve tehlikeli durumları bildirme	Küçük görünse bile ramak kala olay, tehlikeli durum veya arızanın sorumlu veya yetkili kişiye bildirilmesini gerektirir.
Aksesuar ve uygun olmayan kıyafet	Atölye uygulamalarında yüzük, sallanan takılar, gevşek kollar veya açıkta kalan uzun saç risk oluşturabilir. Bu nedenle uygulamaya uygun kıyafet tercih edilmelidir.
Kişisel koruyucu donanım (KKD) kullanımı	İşin gerektirdiği maske, eldiven, gözlük, kulak koruyucu gibi donanımlar doğru şekilde kullanılmalıdır.
Yetkisiz müdahaleden kaçınma	Elektrik bağlantıları, cihaz içi parçalar veya arızalı ekipman öğretmen izni olmadan kurcalanmamalıdır.
Dikkat dağıtıcı davranışlardan kaçınma	Şakalaşma, acelecilik ve dikkatsizlik kazaya neden olabilir, çalışma sırasında dikkat korunmalıdır.

SIRA SİZDE

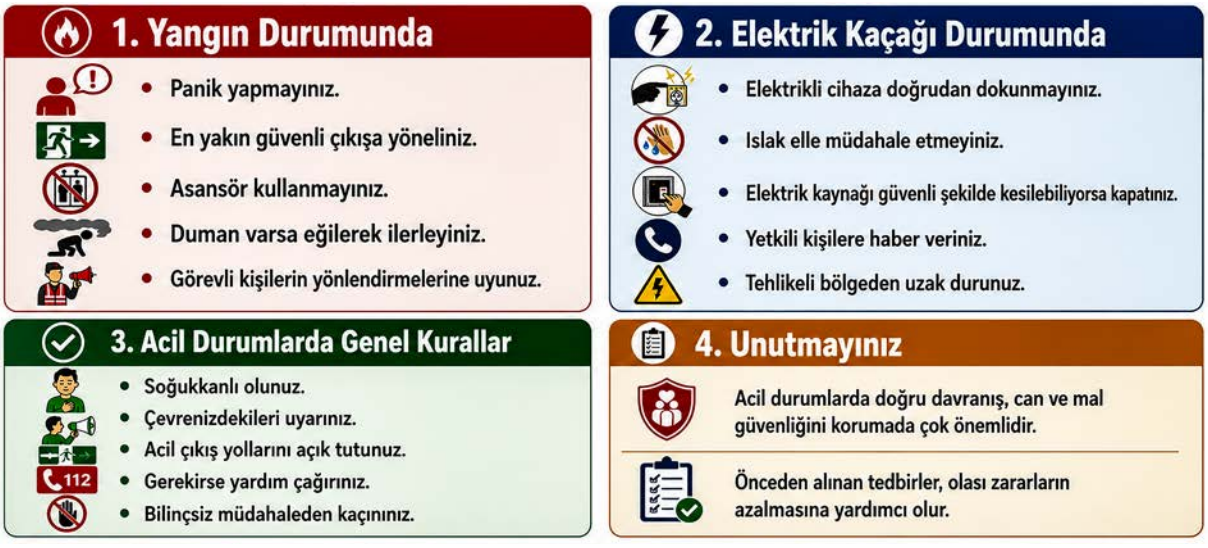
Aşağıdaki Koruma Yönetimi ve KKD Eşleştirme Tablosu'na göre önce ortamda alınabilecek güvenlik tedbirini, gerekiyorsa uygun kişisel koruyucu donanımı yazınız. Seçiminizi kısa bir gerekçeyle açıklayınız.

Durum	Önce Alınacak Ortam / Toplu Koruma Tedbiri	Gerekli KKD	Gerekçe
1. Tozlanmış bilgisayar kasasının temizlenmesi			
2. Yüksek ses çıkaran cihazların yanında çalışma			
3. Temizlik maddesiyle yüzey temizliği yapılması			
4. Küçük parça veya toz sıçrama ihtimali olan işlem			

Acil Durumlarda Doğru Hareket Etme

Acil durumlarda paniğe kapılmadan, kurallara uygun hareket etmek gerekir (Görsel 1.7). Yangın, elektrik kaynaklı tehlike, cihazdan duman çıkması, yaralanma gibi durumlarda kişi öncelikle kendi güvenliğini sağlamalı ve durumu sorumlu ya da görevli kişiye bildirmelidir.

Elektrik kaynaklı durumlarda bilinçsiz müdahale yapılmamalı, elektrikli cihaza çıplak elle dokunulmamalı ve güvenli uzaklık korunmalıdır. Yangın durumunda tahliye yönlendirmeleri izlenmeli, çıkış yolları kullanılmalı ve toplanma alanına gidilmelidir. Acil durumlarda çıkış yolları kapatılmamalı, koşuşturmadan ve itişmeden kaçınılmalı, öğretmen veya görevli kişilerin yönlendirmelerine uyulmalıdır.



Görsel 1.7: Acil durumlarda doğru davranış örnekleri

Tahliye, İlk Yardım ve Bildirim Farkındalığı

Tahliye planı, acil durumda binadan güvenli biçimde çıkmak için izlenecek yolu gösterir. Çalışma ortamındaki kişilerin laboratuvar, koridor, acil çıkış kapısı ve toplanma alanı arasındaki ilişkiyi bilmesi gerekir. Tahliye sırasında en yakın güvenli çıkış kullanılmalı, asansör kullanılmamalı ve toplanma alanına gidildikten sonra öğretmenin yönlendirmesi beklenmelidir.

İlk yardım uygulamaları, bu konuda eğitim almış kişiler tarafından yapılmalıdır. Öğrenci, kaza anında bilinçsiz müdahale etmek yerine durumu öğretmene veya görevli kişiye bildirmeli, çevrenin güvenliğine dikkat etmeli ve yetkililerin yönlendirmesini beklemelidir.

BİLGİ KUTUSU

Bilişim laboratuvarlarında bilgisayar parçalarıyla çalışırken yalnızca görünen tehlikelere dikkat etmek yeterli değildir. İnsan vücudunda biriken **statik elektrik**, RAM, anakart, ekran kartı ve depolama birimi gibi hassas elektronik parçalara zarar verebilir. Bu zarar bazen hemen fark edilmez; parça daha sonra hatalı çalışabilir veya hiç çalışmayabilir.

Bu nedenle donanım montajı ve bakım işlemlerinde **antistatik bileklik**, antistatik mat veya uygun çalışma yüzeyi kullanılmalıdır. Elektronik parçalara gereksiz yere dokunulmamalı, parçalar kenarlarından tutulmalı ve işlem öğretmen ya da yetkili kişi yönergesine göre yapılmalıdır. Antistatik önlem almak, hem cihaz güvenliğini hem de güvenli çalışma alışkanlığını destekler.

4. ETKİNLİK: Güvenli Çalışma Bilgi Görseli Hazırlıyorum

Süre: 40 dakika

Amaç: Acil durumlarda doğru davranışları, uygun güvenlik işaretlerini ve gerekli kişisel koruyucu donanımları ilişkilendirerek açıklamak

Yönerge:

1. Adım: Aşağıdaki tabloyu inceleyiniz. Yangın, elektrik kaynaklı tehlike, cihazdan duman çıkması, kimyasal temizlik maddesiyle temas, yoğun toz oluşumu, yüksek sesli cihaz kullanımı gibi durumlardan birini seçiniz.

2. Adım: Seçtiğiniz durum için tehlikeyi, olası riski, doğru davranışı, uygun güvenlik işaretini ve gerekiyorsa kullanılacak kişisel koruyucu donanımı belirleyiniz.

3. Adım: Bilgi görselinizde dikkat çekici özgün bir uyarı mesajı, doğru renkte işaret kullanımı ve kısa açıklamalar yer almalıdır.

Bölüm	Açıklama
Seçilen Durum	Yangın, elektrik kaynaklı tehlike, toz, gürültü, kimyasal temas vb. durumlar
Tehlike ve Olası Risk	Bu durumun neden tehlikeli olduğunu ve yol açabileceği sonuçları yazınız.
Doğru Davranış	Kişinin yapması veya yapmaması gereken güvenli davranışları belirtiniz.
Güvenlik İşareti ve Renk	Kullanılabilecek işaret türünü ve rengini yazınız.
Uygun Kişisel Koruyucu Donanım	Gerekiyorsa kullanılacak koruyucu donanımı belirtiniz.
Özgün Uyarı Mesajı	Kısa, dikkat çekici ve hedef kitleye uygun bir güvenlik mesajı yazınız.

Değerlendirme:

a) Çalışmanız öğretmeniniz tarafından **Kontrol Listesi** ile değerlendirilecektir. Çalışmanızı planlarken kontrol listesinde yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak kontrol listesine ulaşabilirsiniz.

b) Çalışmanızı tamamladığınızda **Akran Değerlendirme Formunu** doldurarak arkadaşınızı değerlendiriniz. Verilen karekodu kullanarak akran değerlendirme formuna ulaşabilirsiniz.

Kontrol listesine ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Akran değerlendirme formuna ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

1.5. MESLEK HASTALIKLARI VE BU HASTALIKLARDAN KORUNMA

Meslek Hastalıkları

Meslek hastalığı, yapılan işin niteliği veya çalışma koşulları nedeniyle zaman içinde ortaya çıkan sağlık sorunudur. İş kazalarından farklı olarak meslek hastalıkları çoğu zaman aniden değil uzun süreli etkiler sonucunda gelişir.

Meslek hastalıklarından korunmak için çalışma ortamındaki riskleri erken fark etmek, uygun çalışma alışkanlıkları geliştirmek ve koruyucu önlemleri düzenli olarak uygulamak gerekir. Ergonomik oturuş, düzenli ara verme, ortamı havalandırma, gürültüden korunma ve gerekli durumlarda kişisel koruyucu donanım kullanma bu önlemler arasında yer alır.

Temel amaç tıbbi tanı koymak değil risk oluşturan çalışma koşullarını fark etmek ve uygun korunma davranışlarını seçmektir.

Meslek Hastalıklarına Yol Açabilecek Başlıca Risk Etkenleri

Bilişim ve laboratuvar ortamlarında meslek hastalıklarına yol açabilecek riskler yalnızca oturuş biçimiyle sınırlı değildir. Toz, gürültü, ekran kullanımı, kimyasal temas, yetersiz havalandırma, tekrarlayan el-bilek hareketleri ve uzun süre aynı pozisyonda çalışma da dikkat edilmesi gereken risk etkenleri arasındadır. Tablo 1.9'da meslek hastalıklarına yol açılabilecek başlıca risk etkenleri listelenmiştir.

Tablo 1.9: Meslek Hastalıklarına Yol Açılabilecek Başlıca Risk Etkenleri

Risk Alanı	Örnek Durum	Korunma Yolu
Solunum Riski	Tozlanmış bilgisayar kasasının temizliği, toner değişimi, yetersiz havalandırma	maske kullanılması, ortamın havalandırılması, toz kaynağına yüzün yaklaştırılmaması
İşitme Riski	Yüksek sesli cihazlar, basınçlı hava spreyi/tüpleri veya uzun süreli fan sesi	gerekli durumlarda kulak koruyucunun kullanılması, gürültü kaynağından uzak durulması
Kas-İskelet Sistemi Riski	Uzun süre aynı duruşta çalışma, uygun olmayan masa ve sandalye kullanımı	ekran yüksekliğinin ayarlanması, doğru oturulması, kısa araların verilmesi
Göz Sağlığı Riski	Uzun süre parlak ekrana bakmak, yetersiz aydınlatma	ekran parlaklığının ayarlanması, gözlerin dinlendirilmesi, uygun aydınlatmanın sağlanması
Cilt ve Kimyasal Temas Riski	Temizlik maddeleri veya toner tozu ile dikkatsiz temas	eldivenin kullanılması, etiket bilgilerine uygun davranılması, öğretmenin yönergesine uyulması
Psikososyal Zorlanma	Uzun süre kesintisiz çalışma, zaman baskısı ve düzensiz görev paylaşımı	çalışmanın planlanması, görev paylaşımının yapılması, gerektiğinde destek istenmesi

BİLGİ KUTUSU

Ekran Karşısında Sağlıklı Çalışma

Programlama, veri girişi, tasarım ve belge hazırlama gibi bilgisayar başında yapılan çalışmalarda gözleri, el-bilekleri ve dikkati korumak önemlidir. Çalışma sırasında kısa aralar verilmeli, ekran parlaklığı ortam ışığına göre ayarlanmalı ve bilekler doğal konumda tutulmalıdır. Gözleri dinlendirmek için yaklaşık her 20 dakikada bir, 20 saniye boyunca uzaktaki bir noktaya bakılması önerilir.

Meslek hastalıklarına neden olabilecek bazı riskler ve korunma yolları Görsel 1.8'de özetlenmiştir.



Görsel 1.8: Meslek hastalıklarına yol açabilecek riskler ve korunma yolları

BİLGİ KUTUSU

Meslek hastalıklarına yol açabilecek riskler, ortaya çıktıkları etkene göre farklı başlıklarda incelenebilir. Bu sınıflandırma, çalışma ortamında hangi risk türüne karşı hangi korunma davranışının seçileceğini anlamaya yardımcı olur.

Tablo 1.10'da meslek hastalıklarına yol açabilecek risk etkenlerinin sınıflandırılması verilmiştir.

Tablo 1.10: Meslek Hastalıklarına Yol Açabilecek Risk Etkenlerinin Sınıflandırılması

Sınıflandırma	Örnek Risk Etkenleri	Korunma Yolları
Fiziksel Etkenler	Gürültü, yetersiz aydınlatma, sıcaklık, hava akımı	Gürültülü cihazların yanında gerekli koruyucuları kullanmak, ekran yansımasını azaltmak ve ortam koşullarını düzenlemek
Kimyasal Etkenler	Temizlik maddeleri, toner/toz, sprey temizleyiciler, gaz veya buhar	Etiketli ürünleri kullanmak, eldiven ve maske takmak, ortamı havalandırmak
Biyolojik Etkenler	Mikroorganizmalar, hijyen yetersizliği, ortak kullanılan yüzeyler	Ortak kullanılan alanlarda temizlik ve hijyen kurallarına uygun davranmak
Ergonomik Etkenler	Uygun olmayan oturuş, tekrarlayan el-bilek hareketleri, uzun süre aynı pozisyonda kalma	Ekran yüksekliğini ayarlamak, kısa aralar vermek, bilek ve sırt duruşuna dikkat etmek
Psikososyal Etkenler	Zaman baskısı, stres, iletişim sorunları, düzensiz görev paylaşımı	Çalışmayı planlamak, görev paylaşımı yapmak, gerektiğinde yardım istemek, olumlu iletişim kurmak

Ergonomik Çalışma ve Sağlıklı Alışkanlıklar

Ergonomi, çalışma ortamının öğrencinin fiziksel özelliklerine ve yaptığı işe uygun düzenlenmesiyle ilgilidir. Ekranın göz hizasına yakın konumlandırılması, sandalyenin uygun yükseklikte ayarlanması, klavye ve farenin rahat kullanılabilecek biçimde yerleştirilmesi ergonomik çalışma için önemlidir.

Uzun süre aynı pozisyonda kalmak, ara vermeden çalışmak ve uygun olmayan duruş alışkanlıkları boyun, sırt, omuz ve bilek bölgelerinde zorlanmaya neden olabilir. Bu nedenle kısa aralar vermek, oturuşu düzeltmek ve çalışma alanını kişiye uygun hâle getirmek sağlıklı çalışma alışkanlığının bir parçasıdır. Tablo 1.11'de ergonomik çalışma için dikkat edilecek durumlar ve sağlıklı çalışma alışkanlığı için yapılması gerekenler listelenmiştir.

Tablo 1.11: Ergonomi Düzenleme

Ergonomik Düzenleme	Dikkat Edilecek Durum	Sağlıklı Çalışma Alışkanlığı
Ekran Yüksekliği	Ekranın çok aşağı veya çok yukarıda olması	Ekranın göz hizasına yakın konumlandırılması
Oturma Düzeni	Sırt desteği olmadan uzun süre oturmak	Sırtın desteklenmesi, ayakların zemine yerleştirilmesi
Klavye ve Fare Kullanımı	Bileğin uzun süre bükülü kalması	Bileğin doğal konumda tutulması
Ara Verme	Uzun süre kesintisiz çalışmak	Belirli aralıklarla kısa molaların verilmesi
Aydınlatma	Ekranda parlama veya yetersiz ışık olması	Işığı ve ekranın parlaklığının uygun şekilde ayarlanması

SIRA SİZDE

Aşağıdaki tabloyu meslek hastalığına yol açabilecek risk etkeni türü, olası sonuç ve korunma yolu bakımından doldurunuz.

Durum	Risk Etkeni Türü	Olası Sonuç	Korunma Yolu
1. Tozlanmış bilgisayar kasasını maskesiz temizlemek			
2. Yüksek sesli cihazları uzun süre koruyucu kullanmadan çalıştırmak			
3. Ekrana çok yakından ve eğilerek çalışmak			
4. Temizlik maddelerini etiketsiz kaplarda bulundurmak			
5. Sürekli zaman baskısı altında mola vermeden çalışmak			

Kontrol, Bilgilendirme ve Korunma Basamakları

Meslek hastalıklarından korunmada çalışma ortamının düzenli kontrol edilmesi, öğrencilerin güvenli davranışlar konusunda bilgilendirilmesi ve risk oluşturan durumların zamanında bildirilmesi gerekir.

Düzenli kontroller, risklerin erken fark edilmesine yardımcı olur. Bilgilendirme ve eğitim ise güvenli davranışların yalnızca ders sırasında değil günlük yaşamda da uygulanmasını destekler. Atölye ve laboratuvarlarda yapılan kısa güvenlik hatırlatmaları, kontrol formları, güvenlik işaretleri ve örnek olay tartışmaları bu süreci güçlendirir.

Meslek hastalıklarından korunmada öncelik, risk etkenini kaynağında azaltmaktır. Bu mümkün olmadığında çalışma ortamı düzenlenir ve gerekli durumlarda kişisel koruyucu donanım kullanılır. Tablo 1.12'de korunma basamakları, açıklama ve örnekleri listelenmiştir.

Tablo 1.12: Korunma Basamakları

Korunma Basamağı	Açıklama	Örnek
Kaynakta Kontrol	Risk oluşturan etkeni ortaya çıktığı yerde azaltmaktır.	toz oluşturan işlemlerin kontrollü yapılması, cihaz bakımının zamanında yaptırılması
Ortam Kontrolü	Çalışma çevresini güvenli ve sağlıklı hâle getirmektir.	havalandırmanın yapılaması, çalışma alanının düzenli tutulması
Kişisel Korunma	Uygun kişisel koruyucu donanımı doğru şekilde kullanmaktır.	maske, gözlük, eldiven veya kulak koruyucunun kullanılması
Bilgilendirme ve İzleme	Riskleri düzenli takip etmek ve güvenli davranışları desteklemektir.	kontrol listesinin kullanılması, tehlikeli durumların öğretmene bildirilmesi



BİLGİ KUTUSU

Acil Durum Akış Sırası

Laboratuvarda cihazdan duman çıkması, yanık kokusu alınması, bataryada şişme görülmesi veya elektrik kaçağı şüphesi oluşması durumunda aşağıdaki sıra izlenmelidir:

1. **Fark et:** Duman, koku, kıvılcım, aşırı ısınma veya olağan dışı ses gibi belirtileri gözlemle.
2. **Uzaklaş:** Cihaza dokunma, cihazı taşımaya çalışma ve güvenli mesafeye geç.
3. **Bildir:** Durumu öğretmene, görevli kişiye veya yetkili personele haber ver.
4. **Müdahaleyi bekle:** Ana şalter, UPS, batarya veya güç bağlantısına yalnızca yetkili kişi müdahale etmelidir.
5. **Tahliye ol:** Gerekli durumda acil çıkış yönlendirmelerini takip ederek sakin biçimde toplanma alanına git.
6. **Yoklamayı bekle:** Toplanma alanında grubundan ayrılma ve öğretmen ya da görevli kişinin yoklama almasını bekle.

5. ETKİNLİK: Meslek Hastalığı Risk Faktörü Bilgi Görseli Hazırlıyorum

Süre: 30 dakika

Amaç: Meslek hastalıklarına yol açabilecek risk faktörlerini sınıflandırmak, olası sonuçlarını açıklamak ve korunma yollarını belirlemek

Yönerge

1. Adım: Aşağıdaki tabloyu inceleyiniz. Solunum riski, işitme riski, kas-iskelet sistemi zorlanması, göz yorgunluğu, kimyasal temas riski veya psikososyal risklerden birini seçiniz.

2. Adım: Seçtiğiniz risk faktörünün fiziksel, kimyasal, biyolojik, ergonomik veya psikososyal etkenlerden hangisiyle ilişkili olduğunu belirleyiniz.

3. Adım: Hazırladığınız bilgi görselinde risk faktörünün nedenlerini, olası sonuçlarını, korunma yollarını ve sorumluluk boyutunu kısa bilgilerle düzenleyiniz. Bilgi görselinizde bilişim atölyesi veya laboratuvar ortamına uygun örnekler kullanınız.

Bölüm	Açıklama
Seçilen Risk Faktörü	Solunum, işitme, ergonomi, göz yorgunluğu, kimyasal temas veya psikososyal risklerden birini seçiniz.
Risk Etkeni Türü	Fiziksel, kimyasal, biyolojik, ergonomik veya psikososyal etkenlerden hangisiyle ilişkilidir?
Bilişim ve Laboratuvar Örneği	Bu riskin okul veya laboratuvar ortamında nasıl ortaya çıkabileceğini yazınız.
Olası Sonuçlar	Riskin zaman içinde doğurabileceği sonucu sade biçimde açıklayınız.
Korunma Basamakları	Kaynakta kontrol, ortam düzeni, bilgilendirme ve uygun KKD öneriniz.
Kısa Uyarı Mesajı	Hedef kitleye uygun, özgün ve anlaşılır bir mesaj yazınız.

Değerlendirme: Çalışmanız öğretmeniniz tarafından **Kontrol Listesi** ile değerlendirilecektir. Çalışmanızı planlarken kontrol listesinde yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak kontrol listesine ulaşabilirsiniz.

Kontrol listesine ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

BİLGİ KUTUSU

El-Bilek Sağlığını Korumak

Bilgisayar başında uzun süre klavye ve fare kullanmak el, bilek ve ön kol bölgesinde zorlanmaya yol açabilir. Özellikle bileğin uzun süre bükülü durması, farenin sıkı kavranması, aynı tuş hareketlerinin sürekli tekrarlanması ve çalışma yüzeyinin uygun yükseklikte olmaması zamanla uyuşma, karıncalanma, ağrı veya güçsüzlük gibi belirtilere neden olabilir.

El-bilek sağlığını korumak için klavye ve fare vücuda yakın konumlandırılmalı, bilekler doğal duruşta tutulmalı ve el bileği masa kenarına sert biçimde dayanmamalıdır. Fare kullanırken yalnızca bileği değil, gerektiğinde ön kolu da hareket ettirmek zorlanmayı azaltır. Uzun süreli çalışmalarda aynı hareketi sürekli tekrar etmek yerine kısa aralıklarla el ve parmakları gevşetmek, sağlıklı çalışma alışkanlığı kazanmaya yardımcı olur. El, bilek veya parmaklarda sürekli ağrı, uyuşma ya da güç kaybı hissedilirse bu durum önemsenmeli ve ilgili kişiye bildirilmelidir.

1.1. PROJE	
Konu	Bilişim Laboratuvarı İSG Rehberini Hazırlıyorum
Kapsanan Konular	İSG kavramları (tehlike, risk, ramak kala), güvenlik işaretleri, acil durum davranışları, kişisel koruyucu donanımlar (KKD) ve ergonomi.
Süre	40+40 dakika
Amaç	Bilişim laboratuvarı İSG rehberi hazırlamak
Çalışma Biçimi	Bireysel

Proje Senaryosu

Okul yönetimi, bilişim laboratuvarlarını kullanmaya yeni başlayacak öğrenciler için bir **güvenlik uyum eğitimi** planlamaktadır. Yönetim, bu kuralların sıkıcı bir liste halinde verilmesi yerine, öğrencilerin ilgisini çekecek, görsel ağırlıklı ve profesyonel bir sunumla anlatılmasını istemektedir. Sizden, laboratuvardaki riskleri ve alınması gereken önlemleri (ergonomi, elektrik güvenliği, tahliye vb.) anlatan kapsamlı bir rehber sunumu hazırlamanız beklenmektedir.

1. Aşama: İçerik Taslağı ve Risk Analizi

Kapsanan Beceri: Tehlike ve risk kavramlarını ilişkilendirme, veri toplama

1. Görev

a) Sunumunuzda yer alacak ana başlıkları belirleyiniz. (Örneğin kavramlar, kurallar, acil durumlar).
b) Laboratuvar ortamına özgü en az üç tehlike (Örneğin kablo dağınıklığı, ısınan kasalar, bozuk priz) ve bu tehlikelerin yaratacağı riskleri bir tabloya listeleyiniz. Bu tabloyu sunumunuzda kullanacaksınız.

2. Aşama: Görsel ve İşaret Seçimi

Kapsanan Beceri: Güvenlik işaretlerini doğru bağlamda kullanma

2. Görev

a) Sunumunuza eklemek üzere üç farklı kategoriden (yasaklayıcı, uyarıcı, zorunlu) güvenlik levhası resmi bulunuz veya çiziniz.
b) Seçtiğiniz her levhanın şeklini, rengini ve bilişim atölyesinde tam olarak nereye asılması gerektiğini not alınız.

3. Aşama: Meslek Hastalıkları ve Ergonomi Bölümü

Kapsanan Beceri: Ergonomik kuralları açıklama, koruyucu tedbir geliştirme

3. Görev

a) Uzun süre yanlış duruşla bilgisayar kullanmanın yol açabileceği iki meslek hastalığını araştırınız.
b) Sunumunuzda öğrencilere **doğru oturuş kurallarını** anlatmak için 90 derece kuralını (dirsek, bel, diz açısı) içeren bir bölüm tasarlayınız.

4. Aşama: Sunumun Tasarımı ve Birleştirilmesi

Kapsanan Beceri: Dijital içerik geliştirme, görsel tasarım kurallarına uyma

4. Görev

a) Seçtiğiniz bir sunum hazırlama programını açarak en az 10 slayttan oluşan sunumunuzu tasarlayınız.
b) Slaytlarınızda 6x6 kuralına (bir slaytta en fazla 6 satır, her satırda en fazla 6 kelime) uymaya dikkat ediniz. Metin yığınları yerine madde işaretleri, tablolar ve şemalar kullanınız.

5. Aşama: Sunumu Sergileme ve Değerlendirme

Kapsanan Beceri: Etkili iletişim

5. Görev

Hazırladığınız sunumu sınıf arkadaşlarınızla paylaşınız.

Değerlendirme

a) Çalışmanız öğretmeniniz tarafından **Proje Derecelendirme Ölçeği** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Proje derecelendirme ölçeğine ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

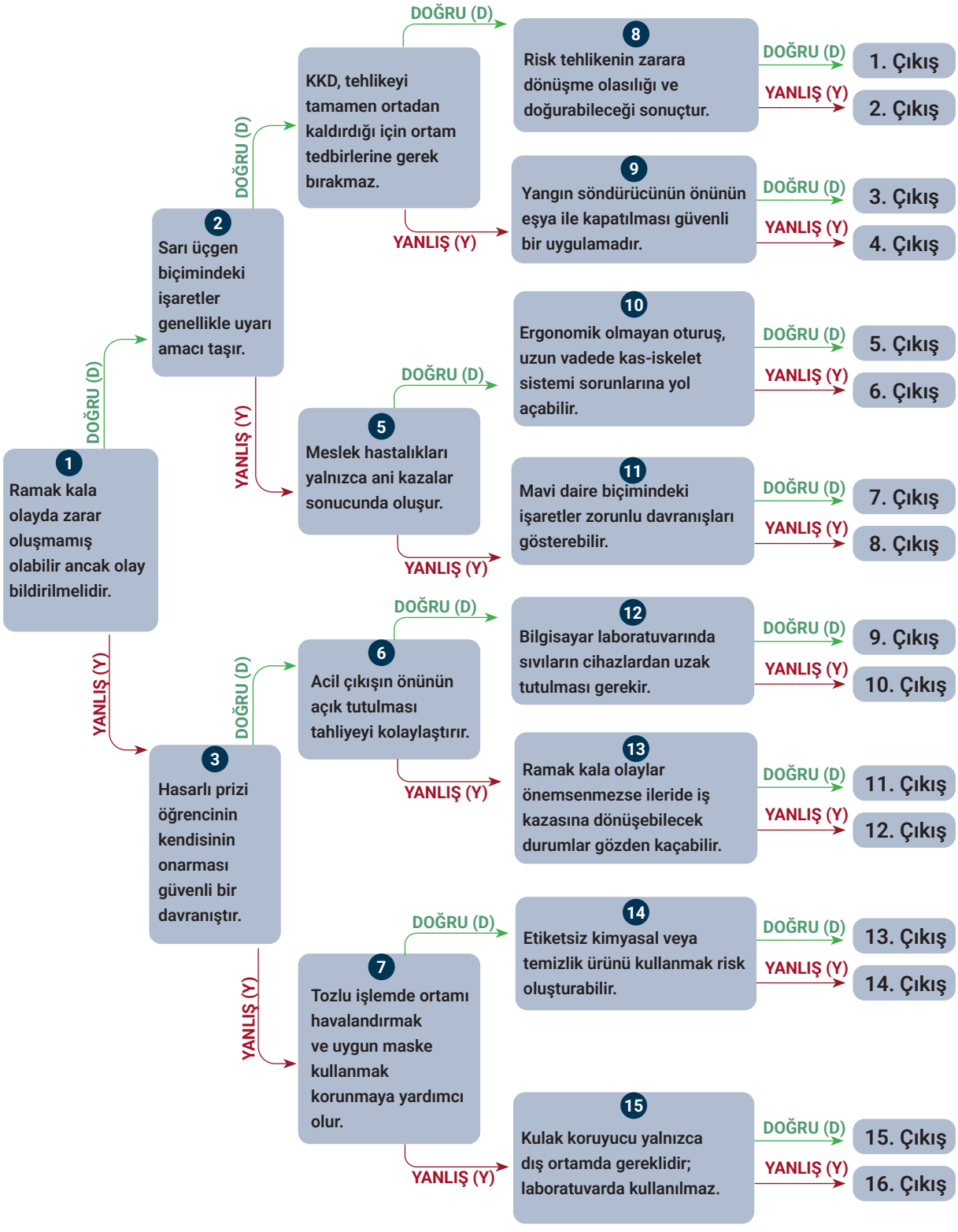
b) **Öz Değerlendirme Formu** proje geliştirme sürecinin sonunda kendinizi değerlendirmeniz amacıyla hazırlanmıştır. Cevaplarınızı tabloda ilgili kutucuğa yazınız. Verilen karekodu kullanarak öz değerlendirme formuna ulaşabilirsiniz.

Öz değerlendirme formuna ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

ÖLÇME VE DEĞERLENDİRME

A) Aşağıda Doğru / Yanlış önermeleri içeren tanılayıcı dallanmış ağaç verilmiştir. Verilen önermenin doğru olduğunu düşünüyorsanız (D), yanlış olduğunu düşünüyorsanız (Y) seçeneğini işaretleyiniz. İlk önermeden başlayarak yapacağınız seçimlerle ve oklar yönünde ilerleyerek bir çıkış noktası belirleyiniz.



B) Aşağıdaki yapılandırılmış gridde numaralandırılmış kutucuklarda iş sağlığı ve güvenliği ile ilgili bazı ifadeler verilmiştir. Kutucuk numaralarını kullanarak soruları cevaplayınız. Aynı kutucuk numarasını birden fazla cevap için kullanabilirsiniz.

(1) Aşırı yüklenmiş çoklu priz	(2) Yangın söndürücünün önünde eşya bulunması	(3) Tozlanmış bilgisayar kasasının maskesiz temizlenmesi
(4) Elektrik tehlikesi uyarı işareti	(5) Acil çıkış yönlendirme işareti	(6) Yüksek sesli cihazın yanında kulak koruyucu kullanmadan çalışma
(7) Maske kullan işareti	(8) Ekran yüksekliğinin uygun şekilde ayarlanması	(9) Temizlik kimyasalının etiketsiz kapta tutulması
(10) Risk kontrol formuna bulgu işlenmesi	(11) İlk yardım noktasının görünür olması	(12) Hasarlı cihazın öğretmene bildirilmesi
(13) Kaygan zeminde düşmeden dengenin sağlanması	(14) Uyarı işaretine rağmen cihaz bağlantısının değiştirilmesi	(15) Basınçlı hava cihazının yüze doğru tutulması
(16) Etiketsiz temizlik ürününün dolaba kaldırılması	(17) Cihaz havalandırma boşluğunun kapatılması	(18) Kısa mola vermeden uzun süre ekrana bakılması

1. Elektrik, yangın, kimyasal veya cihaz kaynaklı riskle ilişkili kutu numaralarını yazınız. (.....)

2. Güvenlik işareti veya acil durum yönlendirmesiyle ilişkili kutu numaralarını yazınız. (.....)

3. Meslek hastalıkları veya uzun süreli sağlık riskleriyle ilişkili kutu numaralarını yazınız. (.....)

4. Güvenli davranış veya risk kontrolüyle ilişkili kutu numaralarını yazınız. (.....)

5. Ramak kala olayları ile ilişkili kutu numaralarını yazınız. (.....)

6. Tehlikeli davranışlar ilişkili kutu numaralarını yazınız. (.....)

7. Tehlikeli durum ilişkili kutu numaralarını yazınız. (.....)

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Aşağıdakilerden hangisi iş sağlığı ve güvenliğinin temel amacını açıklar?

A) Kazadan sonra yalnızca rapor hazırlamak
B) Çalışma ortamında kazaları ve sağlık risklerini önlemeye çalışmak
C) Tüm sorumluluğu yalnızca çalışana bırakmak
D) Laboratuvar düzenini yalnızca estetik açıdan sağlamak
E) Ortamdaki tehlikeleri gidermek yerine yalnızca koruyucu donanım dağıtmak

2. Bir bilgisayar laboratuvarında hasarlı priz fark eden öğrencinin öncelikle yapması gereken davranış hangisidir?

A) Prizi kullanmaya devam etmek
B) Prizi kendisi sökerek onarmak
C) Durumu öğretmene veya yetkili kişiye bildirmek
D) Prizin üzerini kapatmak
E) Laboratuvarın ana şalterini indirerek tüm sistemin elektriğini kesmek

3. Aşağıdakilerden hangisi meslek hastalıklarından korunmaya yönelik bir davranıştır?

A) Uzun süre ara vermeden aynı pozisyonda çalışmak
B) Tozlu ortamda koruyucu kullanmadan çalışmak
C) Ekrandaki detayları daha net görebilmek için ortamdaki tüm ışıkları kapatarak çalışmak
D) Yüksek sesli cihaz yanında kulak koruyucusuz beklemek
E) Ekran ve oturma düzenini uygun hâle getirmek

4. Sarı üçgen biçimindeki bir güvenlik işareti aşağıdakilerden hangi durumu anlatır?

A) Uyarı veya dikkat gerektiren tehlikeyi
B) Zorunlu davranışı
C) Acil çıkış yönünü
D) Yangın ekipmanının yerini
E) Çalışma ortamındaki ilk yardım dolabının yerini

5. Bilişim atölyesinde toner değişimi sırasında yoğun toz oluşuyorsa hangi koruyucu davranış daha uygundur?

A) İşlemi yüzünü yaklaştırarak yapmak
B) Toneri açıkta bırakmak
C) İşlem sırasında yiyecek ve içecek bulundurmak
D) Maske kullanmak ve ortamı havalandırmak
E) Tozu dağıtmak için çalışma masasında doğrudan vantilatör çalıştırmak

6. Aşağıdakilerden hangisi ramak kala olaya örnektir?

A) Öğrencinin elektrik çarpması sonucu yaralanması
B) Raf kenarında duran küçük bir cihazın düşmek üzereyken fark edilip güvenli yere alınması
C) Uzun süre yanlış oturmaya bağlı sırt ağrısı oluşması
D) Yangın söndürücünün görünür yerde bulunması
E) Bilgisayar kasasını tamir ederken elini tornavidayla hafifçe çizmesi

Ç) 1-4. soruları aşağıdaki metne göre cevaplayınız.

Bir bilişim atölyesinde yazıcı toner değişimi yapılmaktadır. Öğrencilerden biri, toner kutusunu açarken eldiven ve maske kullanmaz. O esnada yazıcının yanında açık bir içecek bulunmaktadır. Öğrenci, işlemin uzun süreceğini düşünerek acil çıkış kapısının önüne sandalye koyup oturur. Aynı ortamda basınçlı hava cihazı gürültüyle çalıştırılmasına rağmen öğrenciler kulak koruyucu herhangi bir ekipman kullanmazlar.

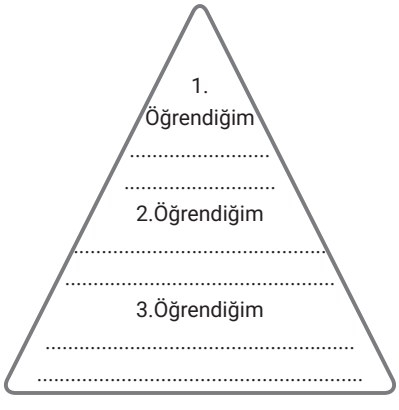
1. Metindeki güvenli olmayan üç durumu yazınız.

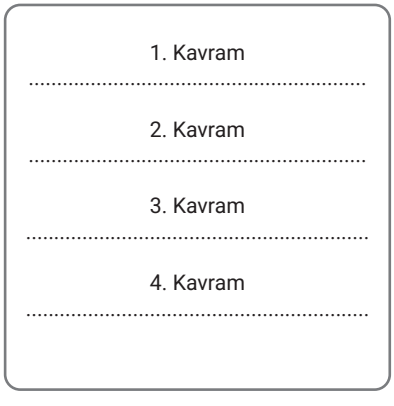
2. Metinde verilen hatalı durumlardan birini seçerek oluşabilecek riski açıklayınız.

3. Seçtiğiniz risk için alınabilecek uygun tedbiri yazınız.

4. Metinde hangi güvenlik işaretleri kullanılabilir? İki örnek yazınız.

D) İş Sağlığı ve Güvenliği öğrenme biriminde öğrendiğiniz üç bilgiyi üçgenin içine, ilk defa karşılaştığınız dört temel kavramı karenin içine ve aklınıza takılan bir soruyu çemberin içine yazınız.









KONULAR

2.1. DİJİTAL PORTFOLYO PLATFORMU ÇEŞİTLERİ

2.2. DİJİTAL PORTFOLYO PLATFORMLARI

ANAHTAR KAVRAMLAR

- depo
- dijital portfolyo
- Markdown
- push
- README
- versiyon geçmişi
- versiyon kontrolü



2. ÖĞRENME BİRİMİ

DİJİTAL PORTFOLYO

BEKLENTİLER

Bu öğrenme biriminde sizden

- dijital portfolyo platformlarının türlerini açıklamanız
- dijital portfolyo platformlarını kullanmanız

beklenmektedir.

HAZIRLIK ÇALIŞMALARI

1. Aşağıda bilgisayarda yapılan çalışmaların saklanması ile ilgili bir metin verilmiştir.

Ahmet bilgisayarında çizdiği dijital resimleri, yaptığı küçük oyunları ve hazırladığı ödevleri farklı klasörlere kaydetmişti. Hayalindeki yazılım kampı için yaptığı başvuruda kendisine: "Bugüne kadar yaptığın çalışmaları bir bağlantı üzerinden gösterebilir misin?" diye soruldu.

Önce duraksadı. Bazı dosyaları eski bilgisayarında kalmış, bazılarını ise hangi klasöre koyduğunu unutmuştu. Elinde harika işler vardı ama onları sunabileceği, "İşte benim yeteneklerim!" diyebileceği düzenli bir vitrini yoktu.

Verilen metne göre aşağıdaki soruları cevaplayınız.

a) Yazılım kampı yetkilisi, Ahmet'in çalışmalarını nasıl bir yöntemle göstermesini istemiştir?

b) Ahmet'in geçmişte yaptığı işlere ulaşmasını imkânsız kılan veya zorlaştıran temel sebepler nelerdir?

2. Üzerinde çok çalıştığınız bir dosya silinir, bozulur ya da eski bir cihazda kaldığı için ulaşılamazsa ne olur?

3. Çalışmalarınızı sergileyebileceğiniz ve dünyanın her yerinden görülebilen bir sayfanız olsaydı, bu sayfada hangi çalışmalarınızın yer almasını isterdiniz?

BAŞLARKEN

Fotoğrafçı, çektiği en iyi kareleri bir albümde toplar. Bu albümle geçmişte yaptığı çalışmaları belgelendirir ve iş görüşmelerinde ya da sergilerde kullanılabilecek bir tanıtım aracı oluşturur. Yazılım dünyasında da benzer bir anlayış vardır. Bir geliştiricinin ne bildiği kadar ne yaptığı da önemlidir. Yazdığı kodları, geliştirdiği projeleri ve bu projelerdeki değişim sürecini düzenli biçimde saklayan ve paylaşan bir geliştirici, dijital ortamda bir öz geçmiş oluşturur.

Dijital portfolyo; bir kişinin dijital ortamda ürettiği çalışmaları, projeleri ve kazandığı becerileri düzenli biçimde bir araya getirdiği çevrim içi koleksiyondur. Kâğıt üzerindeki öz geçmişten farklı olarak dijital portfolyo; çalışmaları doğrudan gösterir, bağlantılar aracılığıyla erişilebilir hâle getirir ve çalışmaların zaman içindeki gelişimi belgeler (Görsel 2.1).



Görsel 2.1: Geleneksel öz geçmiş ve dijital portfolyo

Dijital portfolyo; aynı zamanda yazılımcının bir profil oluşturması, zaman içindeki gelişimini takip etmesi, projeler üzerinde yeniden çalışması ve diğer geliştiricilerle iş birliği yapması için de kullanılır. Portfolyo kavramları Tablo 2.1'de verilmiştir.

Tablo 2.1: Portfolyo Kavramları ve Açıklamaları

Portfolyo Kavramları	Açıklama
Portfolyo	Kişinin ürettiği çalışmaların derlendiği koleksiyon
Dijital Portfolyo	Koleksiyonun çevrim içi ortamda yayımlanmış hâli
Versiyon Geçmişi (Commit)	Projenin zaman içindeki değişim kayıtları
Profil	Platformdaki kişisel kimlik ve çalışmaları gösteren sayfa

BİLGİ KUTUSU

Yazılım sektöründe **yazdığın kodu göster** kültürü yaygınlaşmaktadır. Bu kültürde önemli olan bir yazılım dilini bildiğini söylemek değil o dille üretilen somut çalışmaları başkalarının inceleyebileceği biçimde yayımlamaktır.

2.1. DİJİTAL PORTFOLYO PLATFORMU ÇEŞİTLERİ

Dijital portfolyo oluşturmak için kullanılan platformlar hedef kitleye ve içerik türüne göre farklılaşır. Bir grafik tasarımcının portfolyo ihtiyacı, bir yazılım geliştiricisininkinden farklıdır. Bu nedenle doğru platformu seçmek portfolyonun etkisini doğrudan belirler.

Dijital portfolyo platformları genel olarak üç ana başlık altında incelenebilir.

Genel Portfolyo Siteleri: Tasarım, fotoğrafçılık, yazılım ve illüstrasyon gibi görsel üretim odaklı alanlara yönelik platformlardır. Görsel ağırlıklı bir sunum anlayışı benimsenmiştir. Kullanıcı, hazır şablonlar aracılığıyla çalışmalarını sergiler.

Kişisel Web Siteleri: Kişinin kendi tasarladığı ve yönettiği çevrim içi alanlardır. Kullanıcıya içerik ve görünüm üzerinde tam kontrol sağlar ancak teknik bilgi gerektirir.

Depo (Repository) Platformları: Yazılım geliştirme alanına özgü platformlardır. Çalışmaları sergilemekle kalmaz. Kodun geçmişini, değişiklik kayıtlarını ve iş birliği sürecini de şeffaf biçimde gösterir. Bu özelliği ile yazılım geliştiriciler için en güçlü portfolyo aracı olarak öne çıkar.

SIRA SİZDE

Aşağıdaki tabloda farklı profillere sahip kullanıcılar verilmiştir. Bu kullanıcıların ihtiyaçlarına en uygun platform türünü seçiniz ve seçiminizi tek bir anahtar kelimeyle (gerekçe) açıklayınız.

	Kullanıcı Profili
	I. Grafik tasarım yarışmasına katılan 12. sınıf öğrencisi
	II. Müşterilerine projelerini göstermek isteyen web geliştirici
	III. Bir ekiple mobil uygulama geliştiren yazılımcı
	IV. İlk yazılım işine başvuran yazılım mühendisi

	En Uygun Platform Türü
a)	Genel Portfolyo Sitesi
b)	Kişisel Web Sitesi
c)	Depo Platformu

Gerekçe

1. ETKİNLİK: Dijital Portfolyo Platformlarını Karşılaştırıyorum

Süre: 40 dakika

Amaç: Dijital portfolyo ve depo platformlarını karşılaştırmak

Yönerge: Verilen adımları takip ederek etkinliği tamamlayınız.

1. Adım: Bir genel portfolyo sitesi, depo platformu ve kişisel web sitesi platformunu inceleyiniz.

2. Adım: Her platform için verilen tabloyu doldurunuz.

3. Adım: Araştırmanızı belirtilen sürede sunum ya da rapor biçiminde; akıcı, düzenli ve teknik terimleri doğru kullanarak hazırlayınız ve sınıfla paylaşınız.

Ölçüt	Genel Portfolyo Sitesi	Depo Platformu	Kişisel Web Sitesi Platformu
İncelediğiniz platformun adını yazınız.			
Ücretsiz mi?			
Kod paylaşılabilir mi?			
Versiyon geçmişi var mı?			
İş birliği özelliği var mı?			
Hedef kitlesi kimdir?			
Bir geliştirici için uygun mu?			

Değerlendirme: Çalışmanız öğretmeniniz tarafından **analitik dereceli puanlama anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

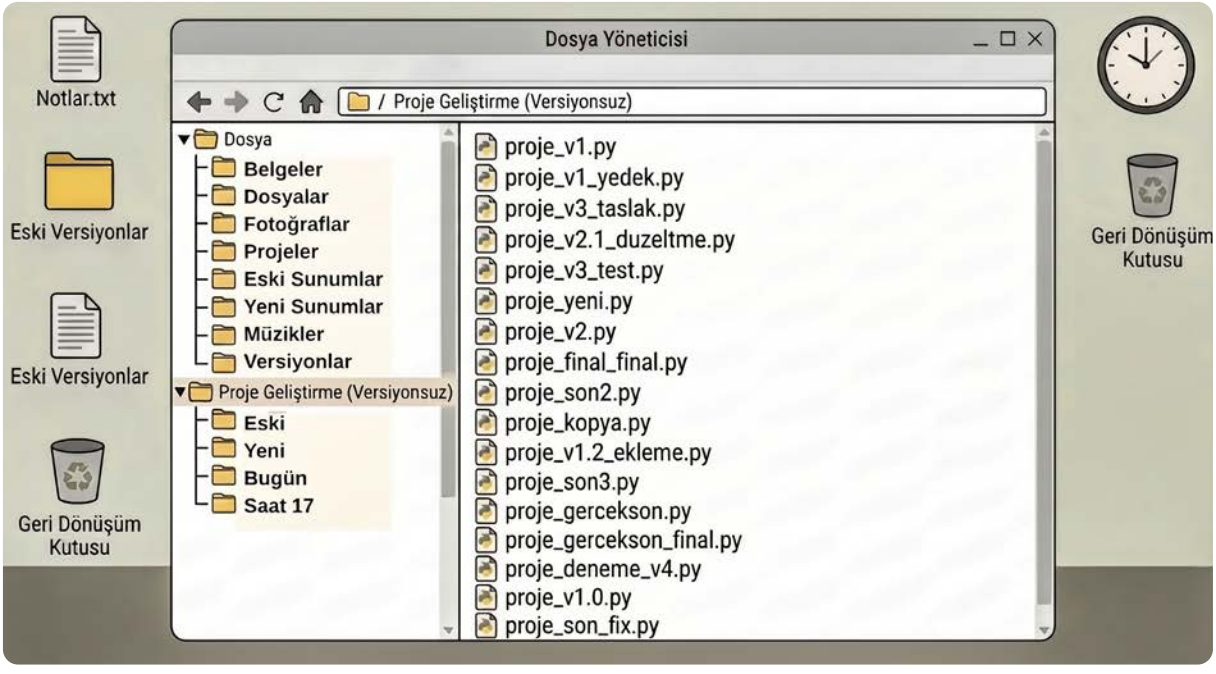
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Depo Platformları

Depo (repository) bir projeye ait tüm dosyaların, klasörlerin ve bu dosyalar üzerinde yapılan değişikliklerin kayıt altına alındığı dijital depolama alanıdır. Depo platformlarında dosyalar saklanabilir. Bu dosyalara kimin, ne zaman, hangi değişikliği yaptığını kaydeder. Bu özellik yazılım geliştirme sürecini şeffaf, izlenebilir ve geri alınabilir kılar.

Versiyon Kontrolü

Bir projeyi geliştirirken yapılan her değişiklik "Bu değişiklik işe yaramazsa eski hâline nasıl dönerim?" gibi yeni bir soruyu beraberinde getirir. Versiyon kontrolü, dosyaların zaman içindeki tüm değişikliklerini kayıt altına alarak bu soruya yanıt verir. Versiyon kontrol sistemi olmadan çalışan bir geliştirici, her güncelleme için farklı bir dosya adı kullanmak zorunda kalır (Görsel 2.2).



Görsel 2.2: Versiyon kontrolsüz yazılım geliştirme

Versiyon kontrol sistemi kullanıldığında tek bir dosya üzerinde çalışılır. Tüm değişiklik geçmişi otomatik olarak kayıt altına alınır ve istenildiğinde önceki herhangi bir sürüme dönülebilir. Tablo 2.2'de versiyon kontrolü ile çalışmanın faydaları görülmektedir.

Tablo 2.2: Versiyon Kontrolsüz ve Versiyon Kontrollü Çalışmanın Karşılaştırılması

Versiyon Kontrolsüz Çalışma	Versiyon Kontrollü Çalışma
Dosyalar elle yeniden adlandırılır.	Değişiklikler otomatik kaydedilir.
Eski sürüme dönmek zordur.	İstenen sürüme kolayca dönülür.
Ekip çalışmasında çakışmalar olur.	Değişiklikler birleştirilebilir.
Kim neyi değiştirdi bilinemez.	Her değişikliğin sahibi kayıtlıdır.

Depo Platformlarının Temel Özellikleri

Depo platformları yalnızca bir bulut depolama alanı olmayıp verilen temel özellikleri bir arada sunar.

- **Versiyon Kontrolü:** Projedeki her değişiklik, kim tarafından ve ne zaman yapıldığı bilgisiyle kayıt altına alınır.
- **İş Birliği:** Farklı geliştiriciler aynı proje üzerinde eş zamanlı çalışabilir; değişiklikler çakışmadan birleştirilebilir.
- **Proje Yönetimi:** Görev listesi, hata bildirimi (issue) ve proje panolarıyla geliştirme süreci takip edilebilir.

Geliştirilen projeler herkese açık depolar aracılığıyla paylaşılabilir. İşverenler ve diğer geliştiriciler bu çalışmaları inceleyebilir.

GitHub dünyanın en yaygın kullanılan depo platformlarından biridir. Açık kaynak projeler için ücretsiz ve herkese açık depolar sunar. Bu platform, geniş topluluk desteği ve iş dünyasındaki yaygınlığı ile yazılım geliştiriciler için dijital bir portfolyo standardına dönüştürür.

GitHub'a benzer özellikler sunan ve yerel bilgisayara kurulabilen (self-hosted) depo platformları da bulunur. Özellikle kurumsal gizlilik gerektiren projelerde tercih edilir. Bu tür depo platformları sürekli entegrasyon ve dağıtım (CI/CD) araçlarını yerleşik olarak barındırır. Açık kaynaklı (open source) ve hafif yapılarıyla öne çıkan bazı depo platformları ise sade ve hızlı arayüzüyle dikkat çeker.

BİLGİ KUTUSU

fork Kavramı: Başka birine ait bir deponun bağımsız bir kopyası kullanıcının hesabında oluşturulur. Böylece orijinal koda zarar vermeden kopya üzerinde özgürce değişiklik yapılarak yeni özellikler test edilebilir.

SIRA SİZDE

Aşağıdaki soruların cevaplarını belirtilen boşluklara yazınız.

1. Bir yazılım projesini versiyon kontrolü olmadan birden fazla kişiyle geliştirmeye çalıştığınızı düşününüz. Hangi sorunlarla karşılaşabilirsiniz?

   .....
   .....
   .....

2. Bazı depo platformlarının self-hosted olması ne anlama gelir ve bu özellik hangi durumlarda avantaj sağlar?

   .....
   .....
   .....

2.2. DİJİTAL PORTFOLYO PLATFORMLARI

GitHub platformunu kullanmaya başlamak için **github.com** adresinden ücretsiz bir hesap oluşturulur (sign up). Hesap oluşturulurken seçilen kullanıcı adı (username) zamanla dijital kimliğin bir parçası olur. Bu nedenle sade, profesyonel ve hatırlanması kolay bir kullanıcı adı seçilmesi önerilir.

BİLGİ KUTUSU

Kullanıcı Adı Seçimi: Seçilen kullanıcı adı, profilin URL'sini github.com/kullanıcıadı şeklinde doğrudan belirler. İleride bu bağlantı bir öz geçmişe veya iş başvurusuna eklenebilir. Bu nedenle gerçek isme yakın, profesyonel bir kullanıcı adı tercih edilmelidir.

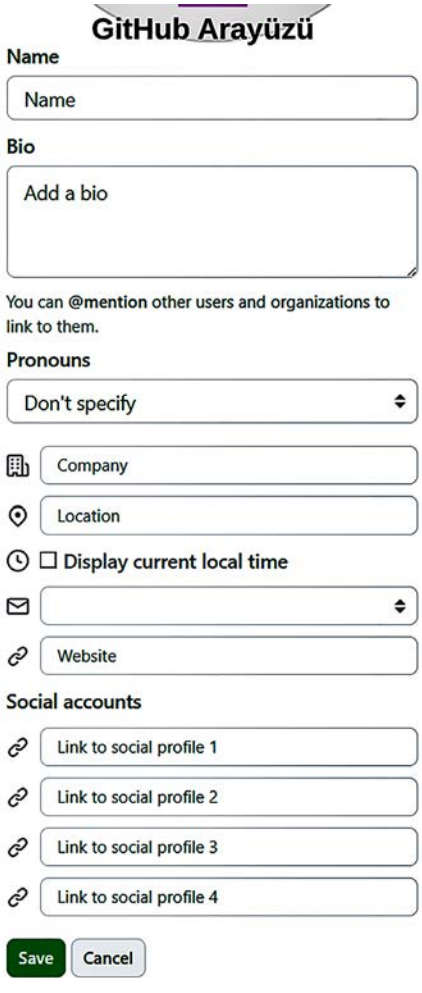
Hesap oluşturulduktan sonra görüntülenen ana sayfa Tablo 2.3'te belirtilen temel bölümlerden oluşur.

Tablo 2.3: Portfolyo Kavramları ve Açıklamaları

Bölüm	Açıklama
Profil (Profile)	Kullanıcı adı, profil fotoğrafı, biyografi ve sabitlenmiş depolar
Depolar (Repositories)	Kullanıcıya ait tüm projelerin listelendiği alan
Akış (Feed)	Takip edilen kullanıcı ve projelerin güncel aktiviteleri
Katkı Grafiği (Contribution Graph)	Yıl boyunca yapılan versiyon geçmişi sayısını gösteren ısı haritası
Yıldızlar (Stars)	Beğenilen ve kaydedilen depolar

SIRA SİZDE

GitHub'da bir hesap oluşturarak arayüzü inceleyiniz ve Görsel 2.3'teki profil bilgilerinizi doldurunuz.



Görsel 2.3: GitHub profil ayarları

Açıklamalar

Name: GitHub profilinizde görünecek adınız

Bio: Kısa bir tanıtım bilgisi

Pronouns: Size ya da ekibinize nasıl hitap edileceği bilgisi

Company: İleride kurmayı hayal ettiğiniz firma ismini yazabilirsiniz.

Location: Şehir ve / veya ülke bilgisi

E-posta: Başka GitHub kullanıcılarının e-posta adresini görmesini isterseniz bu alandan seçebilirsiniz.

Website: Varsa web site adresiniz

Social Accounts: Sosyal medya hesaplarınız

Doğrudan hesabınızın görüntülendiği URL'yi yazabilirsiniz.

Tüm değişiklikleri yaptıktan sonra **Kaydet** (Save) butonuna basmayı unutmayınız.

Depo Oluşturma ve Dosya Yükleme (Web Arayüzü)

GitHub'da yeni bir depo oluşturmak için ana sayfanın sağ üst köşesindeki "+" simgesine tıklanarak "New repository" seçeneği seçilir.

Yeni bir depo oluştururken Tablo 2.4'teki açıklamalar dikkate alınmalıdır.

Tablo 2.4: Depo Oluşturma Kavramları ve Açıklamaları

Depo Kavramları	Açıklama
Repository Name	Deponun adı yazılır, boşluk yerine tire (-) kullanılır.
Description	Projeyi kısaca açıklar (isteğe bağlı alan).
Public / Private	Herkese açık veya sadece sahibi görebilir.
Add a README file	İşaretlenirse otomatik README.md oluşturulur.

Depo oluşturulduktan sonra web arayüzü üzerinden dosya yüklemek için **"Add file / Upload files"** seçeneği kullanılır. Dosyalar sürükle bırak yöntemiyle ya da dosya seçici ile eklenir. Yükleme tamamlandığında bir **versiyon geçmişi mesajı** yazılarak değişiklikler kaydedilir.

1. ÖRNEK UYGULAMA

Amaç: GitHub web arayüzünden bir depo oluşturarak depoya dosya yerleştirmek

1. Adım: ilk-depom adında herkese açık (public) yeni bir depo oluşturulur. README dosyası ekleme (off) seçeneği işaretlenir.

2. Adım: Bilgisayarda merhaba.txt adında bir metin dosyası oluşturulur. İçine ad ve bu derste öğrenmek istenen bir konu yazılır.

3. Adım: Dosya web arayüzü üzerinden depoya yüklenir. Versiyon geçmişi mesajı olarak "İlk dosyamı ekledim" yazılır. **Değişiklikleri Uygula** (Commit Changes) butonuna basılır.

Git Komut Satırıyla Çalışma

Web arayüzü tek bir dosya yüklemek için pratik olsa da gerçek yazılım geliştirme sürecinde projeler **komut satırı** (terminal) aracılığıyla yönetilir. Bu yöntemde bilgisayardaki yerel (local) proje klasörü ile GitHub'daki uzak (remote) depo birbirine bağlanır; değişiklikler komutlarla kaydedilip gönderilir.

Dosya gönderme sürecini yönetmek için kullanılan araca **Git** adı verilir. Git bir versiyon kontrol sistemidir. GitHub ise Git depolarını çevrim içi olarak barındıran platformdur (Görsel 2.4).



Görsel 2.4: Yerel dosyaları GitHub'a gönderme aşamaları

Git komut satırı aracı ile ilgili temel kavramlar ve açıklamaları Tablo 2.5'te belirtilmiştir.

Tablo 2.5: Temel Git Komut Satırı Aracı Kavramları

Kavram	Açıklama
Git	Yerel bilgisayarda çalışan versiyon kontrol aracı
GitHub	Git depolarını bulutta barındıran platform
Yerel Depo (Local)	Bilgisayardaki proje klasörü
Uzak Depo (Remote)	GitHub'daki çevrim içi depo
Versiyon Geçmişi (Commit)	Yapılan değişikliklerin bir mesajla kaydedilmesi
Dosya Gönderme (Push)	Yerel değişikliklerin uzak depoya gönderilmesi

BİLGİ KUTUSU

Git Aracı Kurulumu: Yerel bilgisayara Git aracını yüklemek için **https://git-scm.com/install/** genel ağ adresinden ilgili işletim sistemi seçilerek kurulum yönergeleri takip edilir.

Dosya gönderim sürecinde kullanılan temel Git komutlarına Tablo 2.6'da yer verilmiştir.

Tablo 2.6: Git Komutları ve Açıklamaları

Komut	Açıklama
git init	Mevcut klasörü bir Git deposuna dönüştürür.
git add .	Klasördeki tüm değişiklikleri bir sonraki yeni versiyon için hazırlar.
git commit -m "mesaj"	Hazırlanan değişiklikleri bir mesajla kaydeder.
git push	Yerel değişiklikleri GitHub'a gönderir.
git remote add origin <url>	Yerel depoyu GitHub'daki uzak depoyla ilişkilendirir.
git status	Hangi dosyaların değiştiğini gösterir.

DİKKAT

Versiyon Geçmişi mesajları önemlidir. Proje geliştikçe üzerinde yapılan değişikliklerin daha kolay anlaşılması için mesajların doğru yazılması gerekir. "güncelleme", "düzeltme" gibi belirsiz mesajlar yerine "Kullanıcı giriş formu eklendi." veya "Hesap makinesi sıfırlama hatası giderildi." gibi açıklayıcı mesajlar yazılmalıdır.

2. ÖRNEK UYGULAMA

Amaç: Bilgisayarda oluşturulan bir metin (txt) dosyasını Git komut satırı aracılığıyla GitHub'a göndermek

Terminal Komutları
mkdir ilk-projem
cd ilk-projem
git init
echo "Merhaba GitHub!" > test.txt
git add .
git commit -m "İlk versiyon: test.txt eklendi"
git remote add origin https://github.com/kullaniciadi/ilk-projem.git
git push -u origin main

BİLGİ KUTUSU

"git add ." ile "git add dosyaadı" Arasındaki Fark: "git add ." komutu klasördeki tüm değiştirilmiş dosyaları bir sonraki versiyon için hazırlar. Yalnızca belirli bir dosya, yeni versiyon için hazırlanacaksa git add dosyaadi.txt biçiminde kullanılabilir. Büyük projelerde yalnızca ilgili dosyaları eklemek, versiyon geçmişini daha anlamlı ve düzenli tutar.

1. UYGULAMA

Yerelden GitHub'a Taşıyorum

Süre: 20 dakika

Amaç: Yerel bilgisayarda bir depo oluşturarak dosyayı GitHub'a aktarmak

Yönerge: Yazılan adımları takip ederek uygulamayı tamamlayınız.

1. Adım: Bilgisayarınızda **portfolyo-projem** adında yeni bir klasör oluşturunuz.

2. Adım: Bu klasörün içine daha önce oluşturduğunuz herhangi bir dosyayı kopyalayınız.

3. Adım: Klasörü git init komutu ile Git deposuna dönüştürünüz.

4. Adım: Dosyayı git add ve git commit komutları ile kaydediniz. Versiyon mesajınızı açıklayıcı yazınız.

5. Adım: GitHub'da **portfolyo-projem** adında yeni bir depo oluşturunuz ve git push komutu ile yerel deponuzu GitHub'a gönderiniz.

Değerlendirme: Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

README.md ile Projeyi Tanıtma

Bir GitHub deposu ziyaret edildiğinde ilk görüntülenen bilgi **README.md** dosyasıdır. Bu dosya projenin vitrinidir. Projenin ne olduğunu, nasıl kullanılacağı ve kimler tarafından geliştirildiğini açıklar. ".md" uzantısı **Markdown** biçimlendirme dilini ifade eder.

Markdown; düz metin ile başlık, liste ve kalın yazı gibi biçimlendirmeler oluşturan basit bir işaretleme dilidir. Karmaşık kod öğrenmek gerekmez. Birkaç sembol ile okunabilir ve düzenli belgeler hazırlanabilir. Tablo 2.7'de en sık kullanılan Markdown etiketleri listelenmiştir.

Tablo 2.7: Temel Markdown Etiketleri

Markdown Söz dizimi	Görünümü
# Başlık 1	En büyük başlık
## Başlık 2	İkinci düzey başlık
****kalın****	Kalın
- madde	• madde
`kod`	Kod

BİLGİ KUTUSU

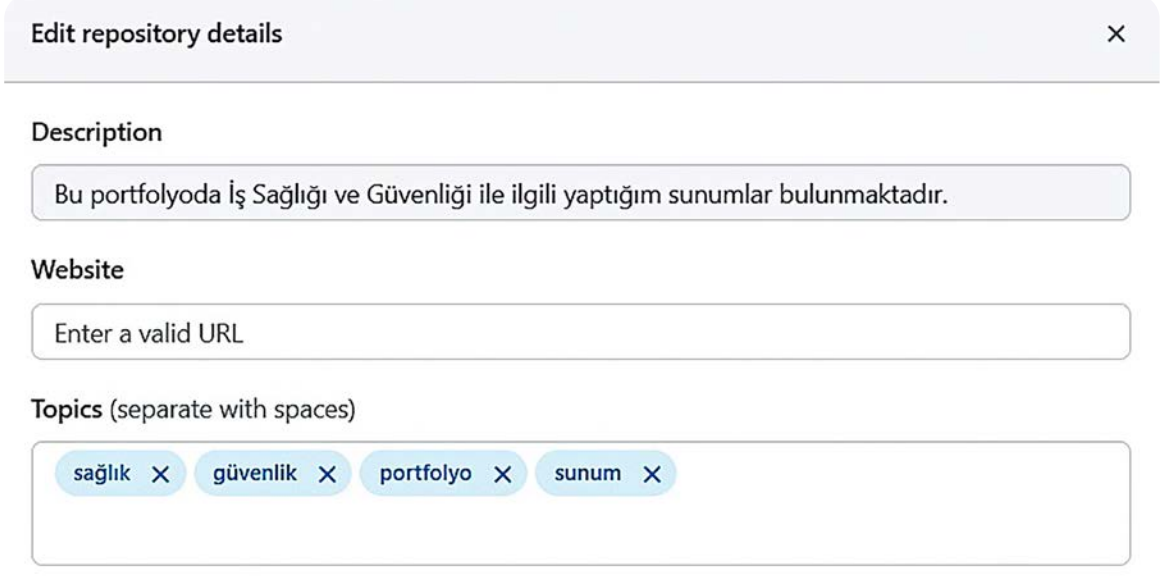
Geliştirilen bir projede ya da oluşturulan bir belgede kullanılan kaynaklar belirtilmelidir.

Basılı bir kitap kaynakçaya **Yazarın Adı Soyadı, Eser Adı, Basımevi, Basım Yeri, Basım Yılı** biçiminde eklenebilir. Örneğin Orhan Şaik GÖKYAY, Bu Vatan Kimin, TCKB, Ankara, 2002

İnternette yer alan bir eğitim içeriği, makale veya bir kod kütüphanesi kaynakçaya **[Kaynak Adı / Başlığı] - [Erişim Bağlantısı (URL)] - [Erişim Tarihi]** biçiminde eklenebilir. Örneğin * [Python Resmi Dokümantasyonu](https://docs.python.org/3/) - Erişim Tarihi: 5 Mart 2026

Depo Açıklamaları ve Etiketler

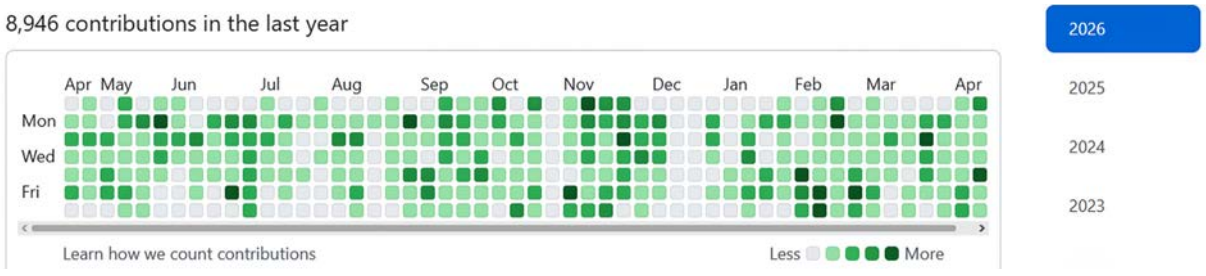
Her depo için kısa bir **açıklama** (description) ve konuyu belirten **etiketler** (topics) eklenebilir. Bu etiketler depoyu arama sonuçlarında öne çıkarır (Görsel 2.5).



Görsel 2.5: Örnek depo açıklaması ve ek bilgiler

Katkı Grafiği

GitHub profilindeki **katkı grafiği** (contribution graph), bir yıl boyunca yapılan güncelleme sayısını günlere göre gösteren ısı haritasıdır. Her kare bir günü temsil eder. O gün ne kadar çok güncelleme yapılmışsa kare o kadar koyu renk alır. Görsel 2.6'daki katkı grafiği ısı haritasında son bir yıl içerisinde 8946 kez depoda güncelleme yapıldığı görülmektedir.



Görsel 2.6: GitHub katkı grafiği ısı haritası

Katkı grafiği yalnızca estetik bir unsur değildir. İşverenler ve diğer geliştiriciler bu grafiğe bakarak bir kişinin ne kadar düzenli çalıştığını, projelere ne ölçüde katkı sağladığını değerlendirir.

DİKKAT

Her güncelleme anlamlı olmalıdır. Katkı grafiğini doldurmak amacıyla anlamsız ve boş güncelleme yapmak, profesyonel çevrelerde olumsuz değerlendirilir. Önemli olan güncelleme sayısı değil her güncellemenin projeye gerçek bir katkı sağlamasıdır.

SIRA SİZDE

Sene boyunca geliştireceğiniz projeleri GitHub'a düzenli olarak yüklemeyi planlıyorsunuz. Bu alışkanlığın size ilerleyen yıllarda nasıl bir avantaj sağlayabileceğini düşününüz. Commit geçmişi olan bir profil ile boş bir profil arasındaki fark, bir işveren için ne ifade eder?

2.1. PROJE	
Konu	Sunum Dosyamı GitHub'a Taşıyorum
Kapsanan Konular	Depo oluşturma, README.md hazırlama, Git komut satırı (init, add, commit, push), güncelleme geçmişi
Süre	40+40+40 dakika
Amaç	Dijital depo yönetimini gerçek bir proje üzerinde uygulamak
Çalışma Biçimi	Bireysel

Proje Senaryosu

Birinci öğrenme biriminde iş sağlığı ve güvenliği kapsamında hazırladığınız "Meslek Hastalıklarına Neden Olabilecek Riskler ve Korunma Yöntemleri" sunum dosyası bilgisayarınızda olduğu için sizden başka kimse bu sunumu göremez ve yorum yapamaz. Bu sebeple dosyayı bilgisayarınızda saklamazsanız ileride bu çalışmayı bulamazsınız. Bu projede sunum dosyanızı; GitHub'a taşıyacak, bir README.md dosyasıyla tanıtacak ve bir değişiklik yaparak yeni hâline güncelleyeceksiniz. Projenin sonunda elinizde hem düzenli bir depo hem de dijital portfolyonuzun ilk gerçek içeriği olacak.

1. Aşama: Depoyu Hazırlamak

Kapsanan Beceri: Depo oluşturma, web arayüzü kullanımı

1. Görev

a) Oluşturacağınız depo bilgilerini verilen tabloya yazınız.

Depo Adı	
Açıklama	
Görünürlük	
README Dosyası	Şimdilik eklenmeyecek, sonraki aşamada komut satırından eklenecek.

b) GitHub hesabınıza giriş yaparak depoyu oluşturunuz ve depo sayfasının ekran görüntüsünü alınız.

2. Aşama: Yerel Klasörü Git'e Bağlamak

Kapsanan Beceri: git init, git remote add origin

2. Görev

a) Bilgisayarınızda sunumun bulunduğu klasörü Git deposuna dönüştürerek GitHub'daki uzak depoyla ilişkilendiriniz.
b) Kullandığınız terminal komutlarını tabloya yazınız.

c) git init komutunu çalıştırdıktan sonra klasörde ne değişti? (Gizli .git klasörünün varlığını kontrol ediniz.)

.....

ç) git remote add origin komutundaki **"origin"** kelimesi ne anlama gelir? Neden bu isim kullanılır?

.....

3. Aşama: README.md Dosyası Hazırlamak

Kapsanan Beceri: Markdown, README yazımı

3. Görev

a) Yerel klasörde bir metin editörü ile sunumunuzu tanıtacak bir README.md dosyası oluşturunuz. Bu dosyada daha önce öğrendiğiniz tüm Markdown söz dizimi ögelerini kullanmanız beklenmektedir.

README.md Dosyasının İçeriği

b) README.md dosyanızda kaç farklı Markdown söz dizimi kullandınız? Verilen tabloyu doldurunuz.

Kullandığınız Markdown Söz dizimi	Yazdığınız Satır
Birinci düzey başlık (#)	
İkinci düzey başlık (##)	
Kalın yazı (**)	
Madde listesi (-)	
Kod stili (`)	

c) README.md dosyasında sunumunuzu oluştururken kullandığınız kaynakları belirttiniz mi?

.....

4. Aşama: Dosyaları Güncellemeye Hazırlamak ve GitHub'a Göndermek

Kapsanan Beceri: git add, git commit, git push

4. Görev

a) Sunum dosyasını ve README.md dosyasını aynı anda ekleyerek ilk güncellemeyi hazırlayınız ve GitHub'a gönderiniz.
Kullandığınız terminal komutlarını tabloya yazınız.

Terminal Komutları

b) git push … komutunu kullandıktan sonra gelen ekran çıktısını yazınız.

.....

c) git add . yerine yalnızca README.md dosyasını eklemek isteseydiniz komutu nasıl yazardınız?

.....

ç) Commit mesajı olarak neden "güncelleme" ya da "düzeltme" yazmak yetersizdir? Kendi commit mesajınızı seçerken nelere dikkat ettiniz?

.....

5. Aşama: Değişiklik Yaparak Yeni Commit Oluşturmak

Kapsanan Beceri: Commit geçmişi, anlamlı mesaj yazımı

5. Görev

a) Gerçek yazılım projelerinde bir dosya hiçbir zaman ilk hâlinde kalmaz, sürekli geliştirilir. Bu aşamada README.md dosyanıza yeni bir bölüm ekleyerek bu değişikliği ayrı bir commit olarak kaydedeceksiniz.
README.md dosyanızı açarak en alta ## Öğrendiklerim alt başlığını açarak bir cümle ile depo güncelleme ile ilgili öğrendiklerinizi yazınız.

Eklenecek Bölüm
Öğrendiklerim

b) Değişikliği kaydediniz ve yeni bir commit oluşturunuz. Kullandığınız terminal komutlarını tabloya yazınız.

Terminal Komutları

c) GitHub'da deponuzun commit geçmişine giderek kaç commit olduğunu doğrulayınız. Her commitin yanında hangi bilgiler görünüyor?

.....

ç) İki commit arasındaki farkı (diff) GitHub arayüzünde nasıl görebilirsiniz? İnceleyiniz ve gözlemlerinizi yazınız.

.....

d) Bir projeyi tek büyük bir commit yerine birden fazla küçük commitlerle kaydetmenin avantajı nedir?

.....

6. Aşama: Depoyu Profilde Sergilemek

Kapsanan Beceri: Profil yönetimi, dijital portfolyo

6. Görev

a) Projeyi tamamladıktan sonra deponuzu GitHub profilinizde sabitleyiniz (pin). Depo sayfasına kısa bir açıklama ve en az bir konu etiketi (topic) ekleyiniz. Eklediğiniz etiketleri yazınız.

.....

b) Deponuzu profilde sabitlediniz mi? Profilinizin bağlantısını aşağıya yazınız.
github.com/.....

c) Deponuzu ziyaret eden biri README.md dosyasını okumadan yalnızca depo sayfasına bakarak projeniz hakkında ne anlayabilir? Açıklama ve etiketlerin önemi nedir?

.....

Değerlendirme

a) Çalışmanız öğretmeniniz tarafından **Proje Derecelendirme Ölçeği** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Proje derecelendirme ölçeğine ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

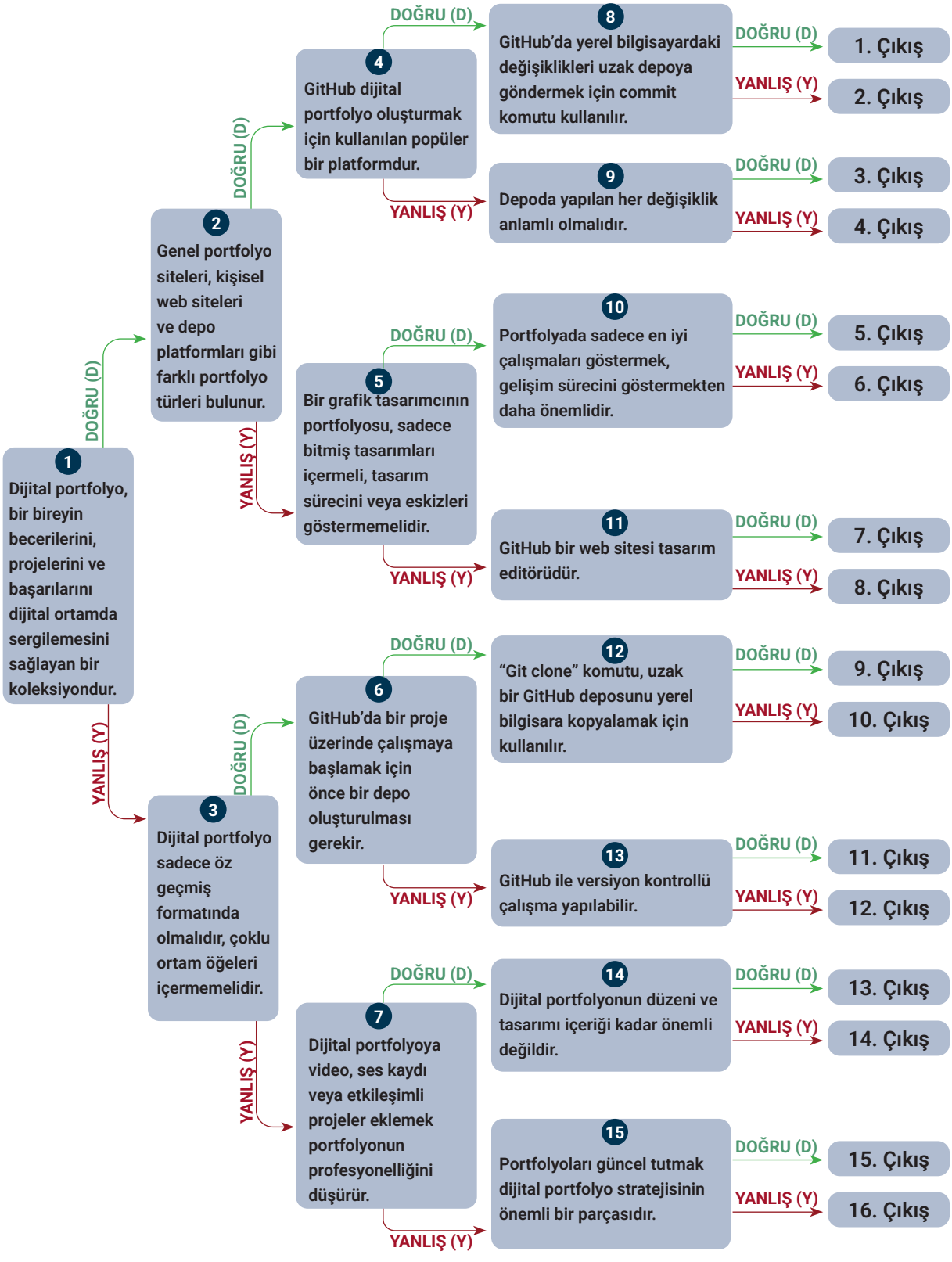
b) **Öz Değerlendirme Formu** proje geliştirme sürecinin sonunda kendinizi değerlendirmeniz amacıyla hazırlanmıştır. Cevaplarınızı tabloda ilgili kutucuğa yazınız. Verilen karekodu kullanarak öz değerlendirme formuna ulaşabilirsiniz.

Öz değerlendirme formuna ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

ÖLÇME VE DEĞERLENDİRME

A) Aşağıda Doğru / Yanlış önermeleri içeren tanılayıcı dallanmış ağaç verilmiştir. Verilen önermenin doğru olduğunu düşünüyorsanız (D), yanlış olduğunu düşünüyorsanız (Y) seçeneğini işaretleyiniz. İlk önermeden başlayarak yapacağınız seçimlerle ve oklar yönünde ilerleyerek bir çıkış noktası belirleyiniz.



B) Aşağıdaki yapılandırılmış gridde numaralandırılmış kutucuklarda dijital portfolyo ile ilgili bazı ifadeler verilmiştir. Kutucuk numaralarını kullanarak soruları cevaplayınız. Aynı kutucuk numarasını birden fazla cevap için kullanabilirsiniz.

(1) profil	(2) sabitleme (pin)	(3) git clone
(4) git push	(5) ** (çift yıldız)	(6) git remote add origin
(7) fork	(8) git init	(9) Markdown

1. Yerel bilgisayarda sıfırdan yeni bir boş Git deposu **başlatmak** ve ardından bu yerel depoyu GitHub üzerindeki uzak depoya **bağlayıp ilişkilendirmek** için sırasıyla hangi komutlar kullanılmalıdır? (.....)

2. GitHub'da başka bir geliştiriciye ait bir deponun bağımsız bir **kopyasını oluşturmak**, sonrasında kodları düzenlemek amacıyla **yerel bilgisayara indirmek** için hangi işlem ve git komutu kullanılır? (.....)

3. Git komut satırında kullanılan komutlar nelerdir? (.....)

4. Depo açıklamalarında kullanılan işaretleme dili nedir ve söz dizimi örnekleri hangileridir? (.....)

5. Dijital portfolyoda belirli depoları öne çıkarmak için hangi özellik kullanılır? (.....)

6. Dijital portfolyo hesabının ana sayfasında doğrudan sergilenen veya yönetilen bileşenler hangileridir? (.....)

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1.

```
git init
git add .
git commit -m "İlk commit"
git remote add origin https://github.com/kullanici/proje.git
git push -u origin main
```

Yukarıdaki komut dizisi çalıştırıldığında aşağıdaki hangi işlem uygulanır?

A) Yalnızca yerel bir Git deposu oluşturulur, GitHub'a herhangi bir şey gönderilmez.
B) Yerel dosyalar Git deposuna eklenir, commit edilir ve GitHub'daki uzak depoya gönderilir.
C) GitHub'daki depo silinir ve yerel dosyalarla yeniden oluşturulur.
D) Dosyalar doğrudan GitHub web arayüzüne yüklenir, Git kullanılmaz.
E) Yalnızca "main" isimli dosya GitHub'a yüklenir, projedeki diğer dosyalar bilgisayarda kalır.

2. Bir GitHub deposunu Public (herkese açık) yapmanın sonucu aşağıdakilerden hangisidir?

A) Yalnızca depo sahibi dosyalara erişebilir.
B) Depo yalnızca davet edilen kullanıcılar tarafından görüntülenebilir.
C) İnternet bağlantısı olan herkes depoyu görüntüleyebilir.
D) Depo yalnızca GitHub hesabı olan kullanıcılar tarafından görüntülenebilir.
E) Depodaki kodlar herkese açık hale gelir ancak projeye ait tüm kullanım hakları GitHub'a devredilmiş olur.

3. Aşağıdaki Markdown söz dizimlerinden hangisi ikinci düzey başlık oluşturur?

A) # Başlık
B) - Başlık
C) **Başlık**
D) ## Başlık
E) [Başlık]

4. Aşağıdakilerden hangisi açıklayıcı ve tercih edilen bir commit mesajı örneğidir?

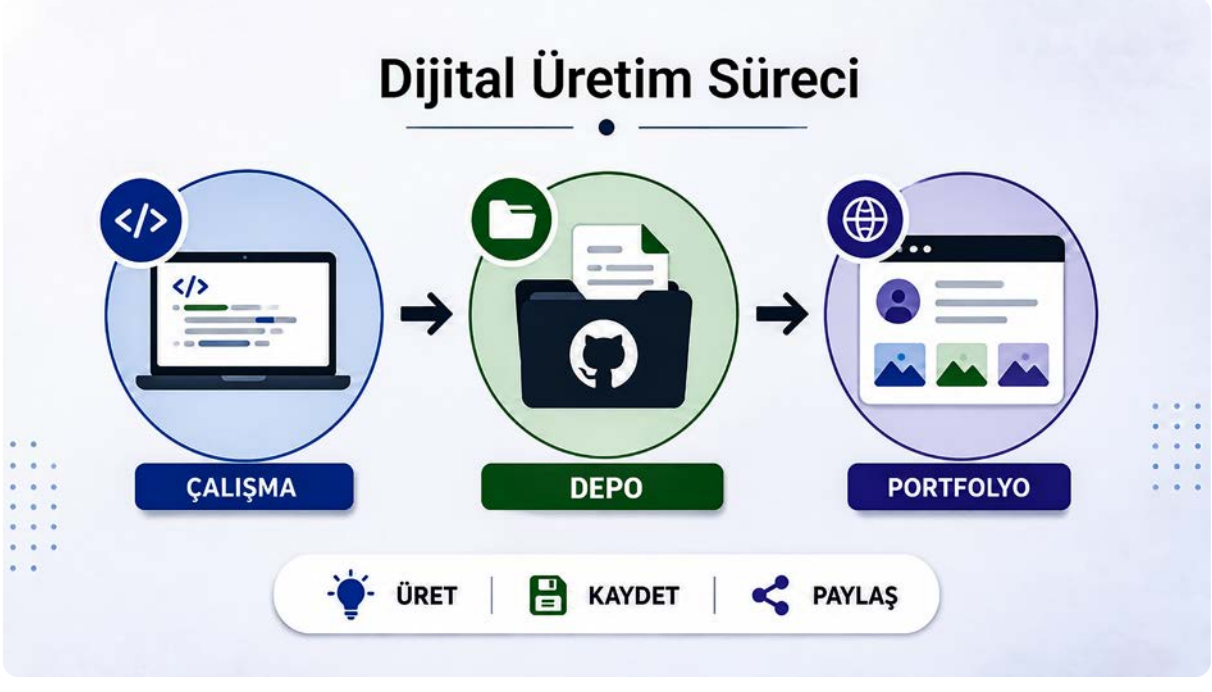
A) "düzeltme"
B) "aaa"
C) "son hali"
D) "Nihayet çalıştı, umarım bir daha bozulmaz!"
E) "Kullanıcı giriş formu eklendi"

5. Yazılım versiyon kontrolü amacıyla kullanılan depo platformlarının ortak özelliği aşağıdakilerden hangisidir?

A) Git tabanlı versiyon kontrolü ve depo barındırma hizmeti sunarlar.
B) Yalnızca mobil uygulama ile kullanılabilirler.
C) Bu platformlar ücretlidir.
D) Tamamı self-hosted (kendi sunucusuna kurulabilir) seçenek sunar.
E) Kod yazmak ve derlemek için geliştirilmiş editör programlarıdır.

Ç) Bu öğrenme biriminde edindiklerinizi değerlendirmek için aşağıdaki çıkış kartını doldurunuz.

Üç Yaz	Dijital portfolyo öğrenme biriminde öğrendiğiniz veya önemli bulduğunuz üç bilgiyi yazınız.
1	
2	
3	
İki Sor	Git komutları ile ilgili merak ettiğiniz veya daha fazla bilgi almak istediğiniz konuyla ilgili iki soru yazınız.
1	
2	
Bir Paylaş	Yıl boyu yapacağınız çalışmalarla oluşturacağınız dijital öz geçmişinizin nasıl görünmesini istediğinizi paylaşınız.





KONULAR

3.1. BİLGİSAYAR PROGRAMLAMA DİLLERİNİN TEMEL YAPISI

3.2. VERİ TİPLERİ

3.3. OPERATÖRLER İLE MATEMATİKSEL VE MANTIKSAL İŞLEMLER

3.4. KOŞULLU İFADELER VE KARAR YAPILARI

3.5. DÖNGÜ YAPILARIYLA TEKRARLI İŞLEMLER

ANAHTAR KAVRAMLAR

- akış şeması
- algoritma
- değişken
- döngü
- karar yapısı
- operatör
- Python
- tür dönüşümü



3. ÖĞRENME BİRİMİ

PROGRAMLAMAYA GİRİŞ

BEKLENTİLER

Bu öğrenme biriminde sizden

- bilgisayar programlama dillerinin temel yapısını açıklamanız
- veri tiplerini kullanmanız
- operatörler ile matematiksel ve mantıksal işlemler yapmanız
- koşullu ifadeler ve karar yapılarını kullanmanız
- döngü yapılarıyla tekrarlı işlemler yapmanız

beklenmektedir.

HAZIRLIK ÇALIŞMALARI

1. Aşağıda bir öğrencinin sabah yaşadığı olayların anlatıldığı bir metin verilmiştir.

Sabah saat yedi. Ertelediğiniz alarm çalıyor ve siz yataktan fırlıyorsunuz. Üzerinize ne giyeceğinizi düşünmeden ilk gördüğünüz kıyafeti alıp giyiyorsunuz. Banyoya koşup dişlerinizi fırçalıyorsunuz. Mutfağa indiğinizde annenizin kahvaltıyı hazırlamadığını görüyorsunuz çünkü o da işe yetişmek için evden erkenden çıkmış. Bir bardak süt içip okul çantanızı kaptığınız gibi kapıdan çıkıyorsunuz. Otobüse binip elinizi cebinize attığınızda telefonunuzun yanınızda olmadığını fark ediyorsunuz. Ardından çantanıza bakıyorsunuz, matematik kitabı da yanınızda değil. "Bugün sınav vardı oysa" diye düşünüyorsunuz. Otobüsten indikten sonra okula ulaşmak için 20 dakika yürüyorsunuz ve sınavı kaçırıyorsunuz. O telaşlı, planlanmamış sabah yüzünden sınavı kaçırdığınız için kendinize çok kızıyorsunuz.

Verilen metne göre aşağıdaki soruları cevaplayınız.

a) Metindeki öğrencinin sabah yaşadığı temel sorun nedir?

b) Öğrenci sabah yaşadığı problemin çözmek için ne yapmalıdır?

2. Sabah hazırlanırken yaptığınız işleri sırasıyla yazınız.

3. Sabah hazırlanırken işleri sıraya koymak, gerekli eşyaları kontrol etmek ve bazı durumlara karar vermek neden önemlidir?

4. Günlük yaptığınız işlerin sıralamasını karıştırmak sizce ne tür sorunlara yol açabilir?

BAŞLARKEN

Hayatta karşılaşılan problemleri çözmek veya bir işi başarıyla tamamlamak için adımları doğru planlamak şarttır. Bir problemi çözmek ya da amaca ulaşmak için tasarlanan sıralı ve mantıksal adımlar bütününe **algoritma** denir.

İnsanlar kahvaltı hazırlamak gibi günlük işlerini yaparken alışkanlıkla ve düşünmeden yapar. Bilgisayar dünyasında ise durum farklıdır. Bilgisayarlar çok hızlı işlem yapabilen araçlar olsa da verilen komutlar olmadan herhangi bir işlemi kendiliğinden gerçekleştiremez. Uygulamanın çalışması, bir oyundaki karakterin zıplaması ya da gönderilen bir mesajın karşı tarafa ulaşması; programcılar tarafından bilgisayara adım adım tanımlanan komutların sonucunda gerçekleşir. Bu adım adım işlem süreci, algoritma kavramının temelini oluşturur.

3.1. BİLGİSAYAR PROGRAMLAMA DİLLERİNİN TEMEL YAPISI

Bir işi belirli bir hedefe ulaştırmak için izlenen sıralı adımların bütününe **algoritma** denir. Algoritma kavramı, ismini Bağdatlı büyük Türk-İslam bilgini Harezmî'den (El-Hârizmî) almıştır.

Bir algoritma hem yazılı hem de görsel olarak da gösterilebilir. Bu görsel gösterime **akış şeması (flowchart)** denir. Akış şemalarında her bir işlem türü için farklı bir sembol kullanılır. Akış şemalarında kullanılan temel semboller Görsel 3.1'de gösterilmiştir.



Görsel 3.1: Akış şemalarında kullanılan temel semboller

Aşağıda, kullanıcıdan iki sayı alıp toplamını ekrana yazdıran basit bir algoritma örneği verilmiştir. Bu algoritmanın akış şeması Görsel 3.2'de gösterilmiştir.

1. Adım: Programı başlat.

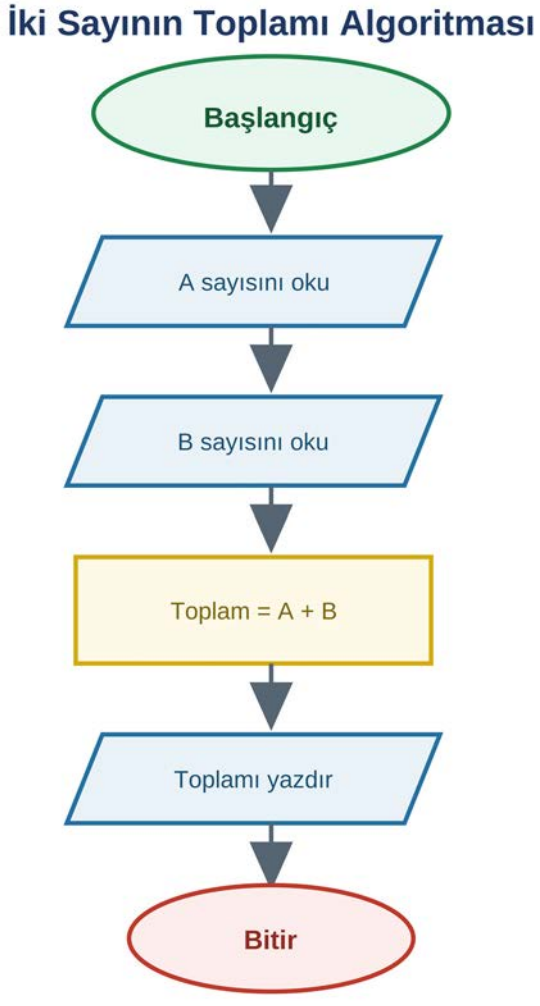
2. Adım: Kullanıcıdan birinci sayıyı al ve A değişkenine ata.

3. Adım: Kullanıcıdan ikinci sayıyı al ve B değişkenine ata.

4. Adım: Toplam = A + B işlemini gerçekleştir.

5. Adım: Toplam değerini ekrana yazdır.

6. Adım: Programı sonlandır.



Görsel 3.2: İki sayının toplamını bulan algoritmanın akış şeması

1. ÖRNEK UYGULAMA

Amaç: Günlük hayattan basit bir algoritma yazmak

1. Adım: Başla.
2. Adım: Su ısıtıcısına su koy.
3. Adım: Suyu kaynat.
4. Adım: Demliğe çay koy.
5. Adım: Kaynayan suyu demliğe ekle.
6. Adım: Çayı 10 dakika demlenmeye bırak.
7. Adım: Çayı bardağa doldur.
8. Adım: Bitir.

Yazılan algoritma adım adım izlendiğinde bir bardak çay elde edilir. İyi bir algoritmanın en önemli özelliği, adımların doğru sırada ve eksiksiz yerine getirilmesidir.

BİLGİ KUTUSU

İyi bir algoritmanın üç temel özelliği vardır: Her adım açık ve anlaşılır olmalı (belirlilik), sonlu sayıda adımda bitmeli (sonluluk) ve aynı girdiler için her zaman aynı sonuç üretilmelidir (kesinlik).

2. ÖRNEK UYGULAMA

Amaç: Bir karar içeren algoritma yazmak

Algoritmaların bazı durumlarında farklı adımlar izlenir. Aşağıdaki algoritma, bir sayının pozitif mi negatif mi olduğunu belirler.

1. Adım: Başla.
2. Adım: Bir sayı al ve S değişkenine ata.
3. Adım: Eğer S >= 0 ise “Pozitif” yaz, değilse “Negatif” yaz.
4. Adım: Bitir.

Bu algoritmanın 3. adımında bir karar verilir. Akış şemasında bu adım, eşkenar dörtgen (karar) sembolü ile gösterilir. Karar adımları, programların farklı durumlara uyum sağlamasını mümkün kılar.

1. UYGULAMA

Günlük Hayattan Bir Problemi Algoritmayla Modelliyorum

Süre: 40 dakika

Amaç: Günlük hayatta karşılaşılan bir problemi algoritma adımları ve akış şeması ile modellemek

Yönerge: Aşağıda günlük hayatta karşılaşılan bir durumun işlem adımları, karışık olarak verilmiştir. Bu adımları inceleyerek algoritma ve akış şemasının temelini açıklayınız.

Problem Durumu

Öğrenci okula gitmeden önce çantasını hazırlarken ders programına uygun kitapların çantasında olup olmadığını kontrol eder ve eksik kitap varsa çantasına ekler.

Karışık İşlem Adımları

- Çantayı kontrol et.
- Başla.
- Eksik kitap var mı?
- Ders programını incele.
- Eksik kitapları çantaya koy.
- Çantanın hazır olduğunu belirt.
- Bitir.

1. Adım: Verilen problem durumunda ulaşılmak istenen amacı açıklayınız.

2. Adım: Karışık verilen işlem adımlarını doğru sıraya koyarak algoritmayı oluşturunuz.

3. Adım: Algoritmadaki başlangıç, işlem, karar ve bitiş adımlarını belirleyiniz.

4. Adım: Her adım için kullanılacak akış şeması sembolünü belirleyiniz.

5. Adım: Belirlediğiniz sembolleri kullanarak akış şemasını tamamlayınız.

6. Adım: Akış şemasında karar sembolünün neden kullanıldığını açıklayınız.

7. Adım: Algoritma ile akış şeması arasındaki ilişkiyi kendi cümlelerinizle açıklayınız.

8. Adım: Hazırladığınız algoritma, akış şeması ve açıklama metnini dijital ortamda düzenleyiniz. Çalışmanızı uygun dosya adıyla kaydediniz ve dijital deponuza yükleyiniz. Depoda, çalışmanın amacını, kullanılan akış şeması sembollerini ve algoritma-akış şeması ilişkisini açıklayan kısa bir README dosyası oluşturunuz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.



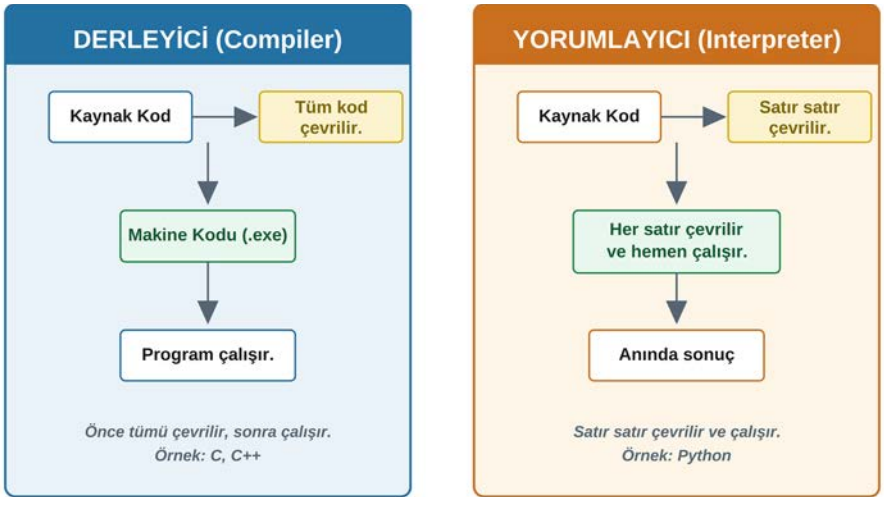
Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



Yorumlayıcı ve Derleyici Programlama Dilleri

Bilgisayarlar yalnızca **makine dilini** (0 ve 1'lerden oluşan diziler) anlar. İnsan diline yakın programlama diliyle yazılan kodların bilgisayar tarafından anlaşılması için makine diline çevrilmesi gerekir. Bu çeviri derleyici ve yorumlayıcı olmak üzere iki yöntemle yapılır. Bu iki yöntemin çalışma prensibi Görsel 3.3'te gösterilmiştir.



Görsel 3.3: Derleyici ve yorumlayıcı dillerin çalışma prensibi

Derleyici (Compiler): Kodun tamamını önce baştan sona okur, makine diline çevirir ve daha sonra çalıştırır. C, C++, Go ve Rust derleyici dillere örnektir.

Yorumlayıcı (Interpreter): Kodu satır satır okur, çevirir ve çalıştırır. Python, JavaScript ve PHP gibi diller yorumlayıcı dillere örnektir.

Derleyici ve yorumlayıcı dillerin karşılaştırması Tablo 3.1'de verilmiştir.

Tablo 3.1: Derleyici ve Yorumlayıcı Programlama Dillerinin Karşılaştırması

Özellik	Derleyici (Compiler)	Yorumlayıcı (Interpreter)
Çalışma Biçimi	Tüm kod önceden çevrilir.	Kod satır satır çevrilir.
Hız	Program derlendikten sonra doğrudan çalıştırıldığı için çoğu durumda yüksek çalışma zamanı performansı gösterebilir.	Kod satır satır yürütüldüğü için bazı durumlarda daha yavaş çalışabilir; ancak performans kullanılan dil, yorumlayıcı ve optimizasyonlara göre değişebilir.
Hata Tespiti	Tüm hatalar derleme sırasında çıkar.	Hatalar çalışma anında çıkar.
Geliştirme Hızı	Her değişiklikte yeniden derleme gerekir.	Anında test etme imkânı vardır.
Örnek Diller	C, C++, Go, Rust	Python, JavaScript, PHP

Tablo 3.1'de görüldüğü gibi her iki yaklaşımın da kendine özgü avantajları vardır. Bu kitapta kullanılan Python, yorumlayıcı bir dildir. Böylece yazılan kod anında test edilebilir.

BİLGİ KUTUSU

Python, sade ve okunabilir söz dizimiyle programlamaya yeni başlayanlar için en çok önerilen dillerden biridir. Yapay zekâ, web geliştirme, veri analizi, oyun programlama gibi birçok alanda kullanılır.

SIRA SİZDE

Derleyici ve yorumlayıcı programlama dillerinin özelliklerine göre aşağıdaki soruları cevaplayınız.

1. Bir programın çok hızlı çalışması gerekiyorsa hangi tür dil tercih edilir? Neden?

2. Bir kodu yazarken sık sık test etmek isteyen bir programcı için hangi tür dil daha uygundur?

3. Python'un hangi türe ait olduğunu ve bu türün sağladığı avantajları açıklayınız.

2. UYGULAMA

Derleyici Dillerin Avantajları Hakkında Sunum Hazırlıyorum

Süre:40 dakika

Amaç: Derleyici dillerin avantajlarını kavramak

Yönerge: Aşağıda derleyici yazılım dilleriyle ilgili bazı bilgi kartları ve inceleme başlıkları verilmiştir. Bu bilgileri kullanarak derleyici yazılım dillerinin avantajlarını açıklayan kısa bir dijital sunum hazırlayınız. Sunumda derleyici dillerin temel özellikleri, avantajları, örnek programlama dilleri ve kullanım alanları yer almalıdır.

Bilgi Kartları

1. Bilgi Kartı: Derleyici Yazılım Dili

Derleyici yazılım dilleriyle yazılan kodlar derleyici tarafından makine diline (çalıştırılabilir dosyaya) dönüştürülür, program ancak bu işlemden sonra çalıştırılır.

2. Bilgi Kartı: Hız Avantajı

Kod makine diline çevrildiği için derleyici dillerle hazırlanan programlar genellikle hızlı çalışır.

3. Bilgi Kartı: Hata Kontrolü

Derleme sırasında kodun tamamı kontrol edilir. Bu durum, bazı hataların program çalışmadan önce fark edilmesini sağlar.

4. Bilgi Kartı: Yaygın Kullanım Alanları

Derleyici diller; işletim sistemleri, oyun motorları, gömülü sistemler, performans gerektiren uygulamalar ve sistem yazılımlarında sıkça kullanılır.

5. Bilgi Kartı: Örnek Diller

C, C++, Go ve Rust derleyici dillere örnek olarak verilebilir.

1. Adım: Verilen bilgi kartlarını inceleyiniz ve derleyici yazılım dillerinin temel özelliğini açıklayınız.

2. Adım: Derleyici yazılım dillerinin avantajlarını hız, hata kontrolü, performans ve kullanım alanı başlıkları altında sınıflandırınız.

3. Adım: C, C++, Go ve Rust dillerinden en az ikisini seçerek bu dillerin hangi alanlarda kullanıldığını araştırınız.

4. Adım: Araştırdığınız bilgileri kısa notlar hâlinde düzenleyiniz ve sunumda hangi sırayla vereceğinizi planlayınız.

5. Adım: Derleyici yazılım dillerinin avantajlarını açıklayan beş ya da yedi slaytlık dijital sunum hazırlayınız.

6. Adım: Sunumunuzda en az bir örnek programlama dili, bir kullanım alanı ve derleyici dillerin en az üç avantajına yer veriniz.

7. Adım: Hazırladığınız sunumu sınıfta paylaşınız ve gelen soruları sunum içeriğine dayanarak cevap veriniz.

8. Adım: Sunum dosyanızı uygun dosya adıyla kaydediniz ve dijital deponuza yükleyiniz. Dosyanın kısa açıklamasını README dosyasına ekleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Geliştirme Ortamının Hazırlanması

Programlamaya başlamadan tıpkı bir marangozun atölyesini ve aletlerini hazırlaması gibi bilgisayara üç temel yazılım kurulması gerekir.

Python Yorumlayıcısı: Yazılan kodu çalıştıran ana programdır. Python'un en güncel kararlı sürümü resmî web sitesinden indirilebilir.

Kod Editörü: Kod yazmayı kolaylaştıran bir programdır. Yazılan kod renklendirilir, hatalar erken gösterilir ve otomatik tamamlama özelliği sunularak iş kolaylığı sağlanır. Birçok ücretsiz kod editörü mevcuttur, öğretmen yönlendirmesiyle uygun bir editör seçilebilir.

Kütüphaneler: Python'un sunduğu hazır kod paketleridir ve ihtiyaç duyulan ek özellikleri kazandırmak için kullanılır. Genellikle **pip** adı verilen komut satırı aracıyla yüklenir.

BİLGİ KUTUSU

Geliştirme ortamı kurulurken kodu çalıştıran yorumlayıcı, kod yazmayı kolaylaştıran editör ve ek özellikler sağlayan kütüphaneler olmak üzere üç temel bileşen bir araya gelir. Bu üçü birlikte, programcının çalışma ortamını oluşturur.

3. UYGULAMA

Python ve Kod Editörünün Kurulumunu Yapıyorum

Süre: 40 dakika

Amaç: Python programlama dili ve kod editörü kurulumunu yaparak çalışma ortamını hazırlamak

Yönerge: Bu uygulamada Python yorumlayıcısı, kod editörü ve temel çalışma ortamı bileşenleri öğretmen rehberliğinde kurunuz . Kurulum işlemleri yapılırken her bileşenin programlama sürecindeki görevini açıklayınız.

1. Adım: Python yorumlayıcısının ne işe yaradığını açıklayınız. Ardından Python'un resmî web sitesinden işletim sisteminize uygun kararlı sürümü belirleyiniz.

2. Adım: Python kurulum dosyasını çalıştırınız. Kurulum sırasında **"Add Python to PATH"** seçeneğini işaretleyiniz. Bu seçeneğin neden gerekli olduğunu kısaca açıklayınız.

3. Adım: Kurulumdan sonra komut satırında **Python --version** komutunu çalıştırınız. Ekranda görülen sürüm bilgisinin ne anlama geldiğini yazınız.

4. Adım: Öğretmen tarafından önerilen kod editörünü kurunuz. Kod editörünün kod yazma sürecindeki görevini açıklayınız.

5. Adım: Kod editöründe Python desteği sağlayan eklentiyi etkinleştiriniz. Kod renklendirme, otomatik tamamlama ve hata gösterimi gibi özelliklerin programcıya nasıl yardımcı olduğunu açıklayınız.

6. Adım: Komut satırında pip install requests komutunu çalıştırınız. pip aracının ve kütüphane yükleme işleminin ne işe yaradığını açıklayınız.

7. Adım: Editörde merhaba.py adlı bir dosya oluşturunuz. Dosyaya print("Merhaba, Dünya!") kodunu yazıp çalıştırınız. print() komutunun görevini açıklayınız.

8. Adım: Aşağıdaki tabloyu doldurarak çalışma ortamındaki temel bileşenlerin görevlerini açıklayınız.

Bileşen	Görevi	Kullanılmadığında Ortaya Çıkabilecek Sorun
Python Yorumlayıcısı		
Kod Editörü		
Python Eklentisi		
pip		
Kütüphane		
print() Komutu		

9. Adım: merhaba.py dosyasını ve doldurduğunuz açıklama tablosunu dijital deponuza yükleyiniz. README dosyasında Python yorumlayıcısı, kod editörü, pip ve kütüphane kavramlarını kısaca açıklayınız.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Kontrol Listesi** ile değerlendirilecektir. Çalışmanızı planlarken kontrol listesinde yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak kontrol listesine ulaşabilirsiniz.



Kontrol listesine ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



3. ÖRNEK UYGULAMA

Amaç: İlk Python programını yazmak

Her programlama dilinde geleneksel olarak yazılan ilk program, ekrana bir karşılama mesajının yazdırıldığı programdır.

```
1   print("Merhaba, Dünya!")
2   print("Python öğrenmeye başlıyorum.")
```

EKRAN ÇIKTISI
Merhaba, Dünya! Python öğrenmeye başlıyorum.

print() fonksiyonuyla parantez içine yazılan bilgi ekrana yazdırılır. Metinler çift veya tek tırnak içinde yazılır.

4. ÖRNEK UYGULAMA

Amaç: Yorum satırı kullanmak

Python'da # işareti ile başlayan satırlar yorum satırı olarak kabul edilir ve program tarafından çalıştırılmaz. Yorum satırları, kodu açıklamak için kullanılır.

```
1   # Bu bir yorum satırıdır, çalıştırılmaz
2   print("Bu satır çalışır")   # Satır sonunda da yorum yazılabilir
3   # print("Bu satır çalışmaz") - yorum hâline getirildi
```

EKRAN ÇIKTISI
Bu satır çalışır

Yorum satırları, kodun ne yaptığını açıklamak ve başkalarının (veya kendinizin daha sonra) kodu anlamasını kolaylaştırmak için kullanılır. Kod yorumlarla açıklanır.

SIRA SİZDE

Adınızı, yaşınızı ve en sevdiğiniz dersi ekrana yazdıran bir program yazınız. Her print() satırının üstüne, o satırın işlevini açıklayan bir yorum satırı ekleyiniz.

3.2. VERİ TİPLERİ

Bilgisayar programları, üzerinde işlem yaptıkları verileri bellekte (RAM) saklar. Programcılar verilerin bellekte saklandığı etiketlenmiş kutulara **değişken** denir. Bir değişkene eşittir (=) işareti ile değer atanır. Bu işleme **atama** denir.

```
yas = 15        # yas adlı değişkene 15 değeri atandı.
ad = "Ayşe"     # ad adlı değişkene "Ayşe" metni atandı.
boy = 1.72      # boy adlı değişkene 1.72 değeri atandı.
```

Değişkenin değeri program çalışırken değiştirilebilir. Program boyunca değiştirilmemesi amaçlanan değerler sabit gibi kullanılır. Python'da sabit tanımlamayı zorunlu kılan özel bir yapı yoktur; bu tür değerler genellikle büyük harflerle yazılan değişken adlarıyla gösterilir. Örnek: PI = 3.14, KDV_ORANI = 0.20

5. ÖRNEK UYGULAMA

Amaç: Bir değişkenin değerini değiştirmek

```
1   puan = 50
2   print("Başlangıç puanı:", puan)
3
4   puan = 75            # değişkenin değeri değiştirildi
5   print("Güncel puan:", puan)
```

EKRAN ÇIKTISI
Başlangıç puanı: 50
Güncel puan: 75

6. ÖRNEK UYGULAMA

Amaç: Birden fazla değişkenle çalışmak

Bir programda birden fazla değişken kullanılabilir. Bir öğrencinin bilgileri aşağıdaki örnekte farklı değişkenlerde saklanır.

```
1   ad = "Zeynep"
2   yas = 16
3   sinif = "10-B"
4   ortalama = 85.5
5
6   print("Öğrenci:", ad)
7   print("Yaş:", yas)
8   print("Sınıf:", sinif)
9   print("Ortalama:", ortalama)
```

EKRAN ÇIKTISI
Öğrenci: Zeynep
Yaş: 16
Sınıf: 10-B
Ortalama: 85.5

Her değişken belleğin ayrı bir bölümünde kendi etiketiyle saklanır. İhtiyaç duyulduğunda bu değişkenlere adlarıyla erişilir.

Değişken İsimlendirme Kuralları

Python'da değişken adları belirli kurallara göre yazılır.

- Bir harf veya alt çizgi (_) ile başlar, sayı ile başlamaz.
- Boşluk, tire (-) veya özel karakter içermez.
- Python dilinde kullanılan anahtar sözcükler (if, for, while, def vb.) değişken adı olarak seçilemez.
- Büyük küçük harfe duyarlıdır: yas ve Yas farklı değişkenlerdir.
- Anlamlı ve açıklayıcı isimler tercih edilir, a yerine ortalama_not kullanılır.
- Birden fazla kelimeden oluşan adlar alt çizgi ile ayrılır (snake_case).

BİLGİ KUTUSU

Geçersiz bir değişken adı kullanmak SyntaxError hatasına yol açar. Örneğin 1sinif (Sayı ile başlıyor.), dogum-tarihi (tire içeriyor) ve for (anahtar sözcük) geçersiz değişken adlarıdır.

7. ÖRNEK UYGULAMA

Amaç: Geçerli ve geçersiz değişken adlarını ayırt etmek

```
# Geçerli değişken adları
ogrenci_no = 101
_ad = "Mehmet"
toplam2 = 50

# Geçersiz değişken adları (hata verir)
# 1sinif = 9        -> sayı ile başlayamaz
# dogum-tarihi = 0  -> tire içeremez
# for = 5           -> anahtar sözcük kullanılamaz
```

Geçerli değişken adlarının harf veya alt çizgi ile başladığı, özel karakterin bulunmadığı görülür. Yorum satırı hâline getirilen geçersiz adlar, neden geçersiz olduklarını göstermek için bırakılmıştır.

4. UYGULAMA

Değişken Tanımlamalarını Değerlendiriyorum

Süre: 20 dakika

Amaç: Değişken isimlendirme kurallarını uygulamak, geçerli ve geçersiz değişken adlarını ayırt etmek, geçersiz olanları kurallara uygun hâle getirmek

Yönerge: Aşağıdaki tabloda verilen her değişken adının geçerli olup olmadığını belirleyiniz. Geçersizse sebebini yazınız ve geçerli alternatif öneriniz.

Değişken Adı	Geçerli mi? (E/H)	Geçersizse Sebep	Doğru Alternatif
kullanici_adi			
3_boyutlu			
kitap-sayisi			
while			
_fiyat			
harclik miktari			
sicaklikDegeri			

Değerlendirme

Çalışmanız öğretmeniniz tarafından **kontrol listesi** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak kontrol listesine ulaşabilirsiniz.

Kontrol listesine ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Temel Veri Tipleri

Python'da en çok kullanılan dört temel veri tipi Tablo 3.2'de verilmiştir. Bir değişkenin tipini öğrenmek için **type()** yerleşik fonksiyonu kullanılır.

Tablo 3.2: Python'da En Sık Kullanılan Temel Veri Tipleri

Veri Tipi	Açıklama	Örnek
int	Tam sayı	yas = 15
float	Ondalıklı sayı	boy = 1.72
str	Karakter dizisi (metin)	ad = "Ayşe"
bool	Mantıksal değer (True/False)	gecti = True

8. ÖRNEK UYGULAMA

Amaç: Değişken tiplerini type() fonksiyonu ile öğrenmek

```
1   ad = "Ahmet"
2   yas = 15
3   boy = 1.72
4   ogrenci = True
5
6   print(type(ad))
7   print(type(yas))
8   print(type(boy))
9   print(type(ogrenci))
```

EKRAN ÇIKTISI
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>

9. ÖRNEK UYGULAMA

Amaç: Aynı veriyi farklı tiplerde saklamak

Bir bilgi, ihtiyaca göre farklı veri tiplerinde saklanabilir. Aşağıdaki örnekte sayı 5'in üç farklı tipte tutulma biçimi gösterilir.

```
1   tam_sayi = 5
2   ondalikli = 5.0
3   metin = "5"
4
5   print(tam_sayi, "->", type(tam_sayi))
6   print(ondalikli, "->", type(ondalikli))
7   print(metin, "->", type(metin))
```

EKRAN ÇIKTISI
5 -> <class 'int'>
5.0 -> <class 'float'>
5 -> <class 'str'>

Üç değişken ekranda "5" gibi görünse de biri tam sayı, biri ondalıklı sayı, biri metin olduğu için bilgisayar açısından farklıdır. metin(str) değişkenleri ile standart aritmetik işlemler (çıkarma, bölme vb.) yapılamaz. Ancak Python'da istisnai olarak metinlerde artı (+) işareti ile birleştirme, yıldız(*) işareti ile de metni tekrarlama işlemi gerçekleştirilebilir.

SIRA SİZDE

Dört farklı tipte değişken tanımlayınız: metin (str), tam sayı (int), ondalıklı sayı (float) ve mantıksal değer (bool). Bu işlemden sonra aşağıdaki soruları cevaplayınız.

1. Her değişkenin hem değerini hem de type() ile tipini ekrana yazdırınız.

2. Her değişkene neden o tipi seçtiğinizi açıklayınız.

print() fonksiyonu ekrana bilgi yazdırmak, input() fonksiyonu ise kullanıcıdan veri almak için kullanılır.

10. ÖRNEK UYGULAMA

Amaç: Kullanıcıdan veri almak ve ekrana yazdırmak

```
1   ad = input("Adınız: ")
2   print("Merhaba", ad, "programlama dünyasına hoş geldiniz!")
```

EKRAN ÇIKTISI
Adınız: Zeynep Merhaba Zeynep programlama dünyasına hoş geldiniz!

BİLGİ KUTUSU

print() fonksiyonunda virgülle ayrılan ifadeler aralarında bir boşluk bırakılarak yazdırılır. Birden fazla değişken ve metin, tek bir print() satırında birlikte kullanılabilir.

Tür Dönüşümü

input() fonksiyonu kullanıcıdan aldığı veriyi her zaman **metin (str)** olarak döndürür. Sayısal işlem yapılacaksa bu veriyi **int()** veya **float()** fonksiyonlarıyla uygun türe dönüştürmek gerekir. Bu işleme **tür dönüşümü (casting)** denir.

11. ÖRNEK UYGULAMA

Amaç: Tür dönüşümünün neden gerekli olduğunu gözlemlemek

```
1   # Tür dönüşümü olmadan
2   yas = input("Yaşınız: ")     # "15" metin olarak alınır
3   # print(yas + 1)             -> HATA! metin ile sayı toplanamaz
4
5   # Tür dönüşümü ile
6   yas = int(input("Yaşınız: "))   # 15 sayıya dönüştürülür
7   print("Gelecek yıl", yas + 1, "yaşında olacaksınız.")
```

EKRAN ÇIKTISI
Yaşınız: 15 Gelecek yıl 16 yaşında olacaksınız.

input() ile alınan veri metin olduğu için, sayısal işlem yapmadan önce **int()** ile sayıya dönüştürülmüştür. Aksi hâlde program TypeError hatası verir.

BİLGİ KUTUSU

Tür dönüşümünün en sık unutulduğu yer input() ile sayı alınan satırlardır. Bir hesaplama yapmadan önce kullanıcıdan alınan verinin int() veya float() ile sayıya dönüştürüldüğünden emin olunuz.

12. ÖRNEK UYGULAMA

Amaç: Farklı türler arasında dönüşüm yapmak

Python'da int(), float() ve str() fonksiyonlarıyla türler arasında dönüşüm yapılabilir.

```
1   # Metni sayıya çevirme
2   metin_sayi = "100"
3   gercek_sayi = int(metin_sayi)
4   print(gercek_sayi + 50)
5
6   # Sayıyı metne çevirme
7   yas = 16
8   yas_metin = str(yas)
9   print("Yaşım: " + yas_metin)
10
11  # Tam sayıyı ondalıklıya çevirme
12  tam = 7
13  ondalikli = float(tam)
14  print(ondalikli)
```

EKRAN ÇIKTISI
150 Yaşım: 16 7.0

SIRA SİZDE

Kullanıcıdan iki sayı alan ve toplamlarını ekrana yazdıran bir program yazınız.

1. input() ile alınan verileri int() ile sayıya dönüştürünüz.

2. Tür dönüşümü yapmazsanız ne olacağını tahmin ediniz ardından deneyerek gözlemleyiniz.

5. UYGULAMA

Farklı Veri Tiplerini Görüntüleyen Uygulama Geliştiriyorum

Süre: 40 dakika

Amaç: Farklı veri tiplerindeki değişkenlerin özelliklerini type() ve print() fonksiyonlarını kullanarak düzenli bir biçimde ekranda görüntülemek

Yönerge: Aşağıdaki adımları takip ederek uygulamayı geliştiriniz.

1. Adım: Editörünüzde veri_tipleri.py adında bir dosya oluşturunuz.

2. Adım: Kendinizle ilgili farklı tipte bilgiler tanımlayınız: ad (str), yaş (int), boy (float), öğrenci olup olmadığı (bool).

3. Adım: Her bir değişken için hem değerini hem de tipini ekrana yazdırınız.

4. Adım: Çıktıyı kontrol ederek her değişkenin doğru tipte olup olmadığını gözlemleyiniz.

5. Adım: Verilen süre içerisinde tüm adımları eksiksiz bir şekilde yerine getirip programı dijital deponuza yükleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.



Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

6. UYGULAMA

Tür Dönüşümü Uygulaması Yazıyorum

Süre: 40 dakika

Amaç: Kullanıcıdan alınan veriye tür dönüşümü işlemini uygulayıp sayısal hesaplamalar yaparak bir sonuç üretmek

Yönerge: Aşağıdaki adımları takip ederek bir alışveriş hesaplama uygulaması geliştiriniz.

1. Adım: Editörünüzde alisveris_hesapla.py adında bir dosya oluşturunuz.

2. Adım: Kullanıcıdan ürünün adı, adedi ve birim fiyatı bilgilerini input() ile alınız.

3. Adım: Adet ve fiyat değişkenlerini int ve float türlerine dönüştürünüz.

4. Adım: Toplam tutarı hesaplayan satırı yazınız.

5. Adım: Sonucu "X adet Y ürünü için toplam tutar: Z TL" biçiminde ekrana yazdırınız.

6. Adım: Programı; beş adet ekmek, iki adet ayran, on adet kalem gibi en az üç farklı veri ile test ediniz.

7. Adım: Programı dijital deponuza yükleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

SIRA SİZDE

1. Kullanıcıdan dikdörtgenin en ve boy bilgilerini alarak alanını ve çevresini hesaplayıp ekrana yazdıran bir program yazınız.
2. Alınan verileri uygun türe dönüştürünüz. Sonuçları anlamlı mesajlarla ekrana yazdırınız.

3.3. OPERATÖRLERLE MATEMATİKSEL VE MANTIKSAL İŞLEMLER

Aritmetik operatörler, sayısal işlemler yapmak için kullanılır. Python'daki aritmetik operatörler Tablo 3.3'te verilmiştir.

Tablo 3.3: Aritmetik Operatörler

Operatör	Anlamı	Örnek	Sonuç
+	Toplama	10 + 3	13
-	Çıkarma	10 - 3	7
*	Çarpma	10 * 3	30
/	Bölme (ondalıklı)	10 / 3	3.333...
//	Tam bölme	10 // 3	3
%	Mod (kalan)	10 % 3	1
**	Üs alma	10 ** 3	1000

13. ÖRNEK UYGULAMA

Amaç: Aritmetik operatörleri kullanmak

```
a = 17
b = 5

print("Toplam:", a + b)
print("Fark:", a - b)
print("Çarpım:", a * b)
print("Bölüm:", a / b)
print("Tam bölüm:", a // b)
print("Kalan:", a % b)
print("Üs:", a ** 2)
```

EKRAN ÇIKTISI
Toplam: 22
Fark: 12
Çarpım: 85
Bölüm: 3.4
Tam bölüm: 3
Kalan: 2
Üs: 289

BİLGİ KUTUSU

Mod operatörü (%) bir sayının başka bir sayıya bölümünden kalanı verir. Bu operatör; programlamada bir sayının başka bir sayıya tam bölünüp bölünmediğini kontrol etmek, zaman/saat dönüşümleri yapmak, döngüsel yapıları sınırlandırmak ve bir sayının tek veya çift olduğunu belirlemek gibi pek çok farklı işlemde temel bir araç olarak kullanılır.

14. ÖRNEK UYGULAMA

Amaç: Mod operatörü ile bir sayının çift/tek olduğunu bulmak

```
1   sayi = 17
2
3   kalan = sayi % 2
4   print(sayi, "sayısının 2'ye bölümünden kalan:", kalan)
5
6   # kalan 0 ise çift, 1 ise tek
7   sayi2 = 24
8   print(sayi2, "% 2 =", sayi2 % 2)
```

EKRAN ÇIKTISI
17 sayısının 2'ye bölümünden kalan: 1
24 % 2 = 0

17 sayısının 2'ye bölümünden kalan 1 olduğu için 17 tek sayıdır. 24 sayısının kalanı 0 olduğu için 24 çift sayıdır. Mod operatörü, bölünebilme kontrolünün temel aracıdır.

15. ÖRNEK UYGULAMA

Amaç: Tam bölme ve normal bölme arasındaki farkı görmek

Python'da iki farklı bölme operatörü bulunur. Aralarındaki fark şu şekilde açıklanır:

```
1   print("17 / 5 =", 17 / 5)      # normal bölme
2   print("17 // 5 =", 17 // 5)    # tam bölme
3   print("17 % 5 =", 17 % 5)      # kalan
```

EKRAN ÇIKTISI
17 / 5 = 3.4
17 // 5 = 3
17 % 5 = 2

Normal bölmede (/) ondalıklı sonuç elde edilir. Tam bölmede (//) ise sonucun yalnızca tam kısmı verilir. 17 = 5 × 3 + 2 olduğu için tam bölüm 3, kalan ise 2'dir.

İşlem Önceliği

Birden fazla operatör aynı ifadede yer alıyorsa, Python belirli bir sıra (öncelik) ile değerlendirir. Sıra matematikteki gibidir ve sıralama şu şekildedir:

1. Parantezler ( )
2. Üs alma **
3. Çarpma, bölme, mod (soldan sağa) *, /, //, %
4. Toplama, çıkarma +, -

16. ÖRNEK UYGULAMA

Amaç: İşlem önceliğinin sonucu nasıl etkilediğini gözlemlemek

```
1   sonuc1 = 2 + 3 * 4
2   print("2 + 3 * 4 =", sonuc1)
3
4   sonuc2 = (2 + 3) * 4
5   print("(2 + 3) * 4 =", sonuc2)
```

EKRAN ÇIKTISI
2 + 3 * 4 = 14
(2 + 3) * 4 = 20

İlk ifadede çarpma önce yapıldığı için sonuç 14'tür. İkinci ifadede ise **parantez** kullanıldığı için önce toplama yapılmış ve sonuç 20 olmuştur. Parantezler, işlem sırasını değiştirmenin en güvenli yoludur.

SIRA SİZDE

Aşağıdaki ifadelerin sonuçlarını önce kâğıt üzerinde hesaplayınız sonra Python ile kontrol ediniz.

1. 10 + 6 / 2
2. (10 + 6) / 2
3. 2 ** 3 + 1
4. 20 % 7 * 2

Karşılaştırma Operatörleri

Karşılaştırma operatörleri iki değeri karşılaştırır ve bir mantıksal değer (True/False) döndürür. Karşılaştırma operatörleri Tablo 3.4'te verilmiştir.

Tablo 3.4: Karşılaştırma Operatörleri

Operatör	Anlamı	Örnek	Sonuç
==	Eşit mi?	5 == 5	True
!=	Eşit değil mi?	5 != 3	True
>	Büyük mü?	5 > 3	True
<	Küçük mü?	5 < 3	False
>=	Büyük veya eşit mi?	5 >= 5	True
<=	Küçük veya eşit mi?	3 <= 5	True

BİLGİ KUTUSU

Atama operatörü (=) ile karşılaştırma operatörü (==) sık karıştırılır. Tek eşittir (=) bir değişkene değer atar, çift eşittir (==) ise iki değerin eşit olup olmadığını kontrol eder. Bu ikisinin karıştırılması mantık hatalarına yol açar.

17. ÖRNEK UYGULAMA

Amaç: Karşılaştırma operatörlerinin sonuçlarını gözlemlemek

```
1   a = 10
2   b = 7
3
4   print("a == b:", a == b)
5   print("a != b:", a != b)
6   print("a > b:", a > b)
7   print("a <= b:", a <= b)
```

EKRAN ÇIKTISI

```
a == b: False
a != b: True
a > b: True
a <= b: False
```

Mantıksal operatörler, birden fazla koşulu birleştirmek için kullanılır. Mantıksal operatörler Tablo 3.5'te verilmiştir.

Tablo 3.5: Mantıksal Operatörler

Operatör	Anlamı	Açıklama
and	VE	Her iki koşul da doğruysa True döndürür.
or	VEYA	Koşullardan en az biri doğruysa True döndürür.
not	DEĞİL	Koşulun tersini döndürür (True ise False).

18. ÖRNEK UYGULAMA

Amaç: Bir sayının hem çift hem 10'dan büyük olduğunu kontrol etmek

```
1   sayi = 14
2
3   cift_mi = (sayi % 2 == 0)
4   buyuk_mu = (sayi > 10)
5
6   print("Sayı çift mi?", cift_mi)
7   print("Sayı 10'dan büyük mü?", buyuk_mu)
8   print("Hem çift hem 10'dan büyük mü?", cift_mi and buyuk_mu)
```

EKRAN ÇIKTISI

```
Sayı çift mi? True
Sayı 10'dan büyük mü? True
Hem çift hem 10'dan büyük mü? True
```

19. ÖRNEK UYGULAMA

Amaç: or ve not operatörlerini kullanmak

```
1   hafta_sonu = True
2   tatil = False
3
4   print("Bugün izinli miyim?", hafta_sonu or tatil)
5   print("Bugün okula gidecek miyim?", not (hafta_sonu or tatil))
```

EKRAN ÇIKTISI

```
Bugün izinli miyim? True
Bugün okula gidecek miyim? False
```

20. ÖRNEK UYGULAMA

Amaç: Karşılaştırma ve mantıksal operatörleri birlikte kullanmak

```
1  not_ortalamasi = 75
2  devamsizlik = 8
3
4  basarili = not_ortalamasi >= 50
5  devam_yeterli = devamsizlik < 10
6
7  print("Notu yeterli mi?", basarili)
8  print("Devamı yeterli mi?", devam_yeterli)
9  print("Sınıfı geçti mi?", basarili and devam_yeterli)
```

EKRAN ÇIKTISI
Notu yeterli mi? True
Devamı yeterli mi? True
Sınıfı geçti mi? True

SIRA SİZDE

Öğrencinin bir seçme sınavına girebilmesi için devamsızlığının on günden az olması ve daha önce proje geliştirmiş olma koşulları belirlenmiştir. Bu iki bilgiyi değişken olarak tanımlayınız ve öğrencinin sınava girip giremeyeceğini kontrol eden program yazınız. Farklı değerlerle test ediniz.

7. UYGULAMA

Aylık Bütçe ve Tasarruf Planlayıcı Geliştiriyorum

Süre: 40 dakika

Amaç: Kullanıcıdan alınan gelir ve harcama bilgileriyle tasarruf miktarını değerlendiren uygulama geliştirmek

Yönerge: Aşağıdaki adımları takip ederek aylık tasarruf planlayıcı uygulamasını geliştiriniz.

1. Adım: Editörünüzde tasarruf_planlayici.py adında bir dosya oluşturunuz.

2. Adım: Kullanıcıdan aylık gelir, kira, fatura, gıda ve diğer harcama bilgilerini alınız. Hepsini sayısal veriye çeviriniz.

3. Adım: Toplam harcamayı ve tasarrufu hesaplayınız.

4. Adım: Toplam harcama, tasarruf miktarı ve tasarruf yüzdesi bilgilerini ekrana yazdırınız.

5. Adım: Mantıksal operatörlerle bir tasarruf değerlendirmesi yapınız. Tasarruf %20'nin üzerindeyse "Tasarruf hedefinizi başardınız!", aksi takdirde "Tasarruf hedefinize ulaşamadınız." mesajını yazdırınız.

6. Adım: Programı en az üç farklı veri seti ile test ediniz.

7. Adım: Programınızı dijital deponuza yükleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

3.4. KOŞULLU İFADELER VE KARAR YAPILARI

Günlük hayatta her an kararlar verilir. "Hava yağmurluysa şemsiye alınır.", "Sürücü belgesi varsa araç kullanılır." Programlarda da farklı durumlara göre farklı işlemler yapmak için karar yapıları kullanılır.

if Yapısı

En basit karar yapısı **if (eğer)** yapısıdır. Bir koşul doğruysa belirli bir kod bloğunu çalıştırır. Koşul yanlışsa bu blok atlanır.

```
if koşul:
    koşul doğruysa çalıştırılacak blok
```

if satırının sonunda **iki nokta (:)** bulunur ve koşula bağlı kod bloğu **girintili** yazılır. Python'da girinti, bir kod bloğunun hangi yapıya ait olduğunu belirtir. Python standartlarında girinti için 4 boşluk veya 1 sekme(tab) tuşu kullanılır.

21. ÖRNEK UYGULAMA

Amaç: Birden fazla if yapısı kullanmak

Bir programda birbirinden bağımsız birden fazla if yapısının olduğu görülür. Her if kendi koşuluyla ayrı ayrı kontrol edilir.

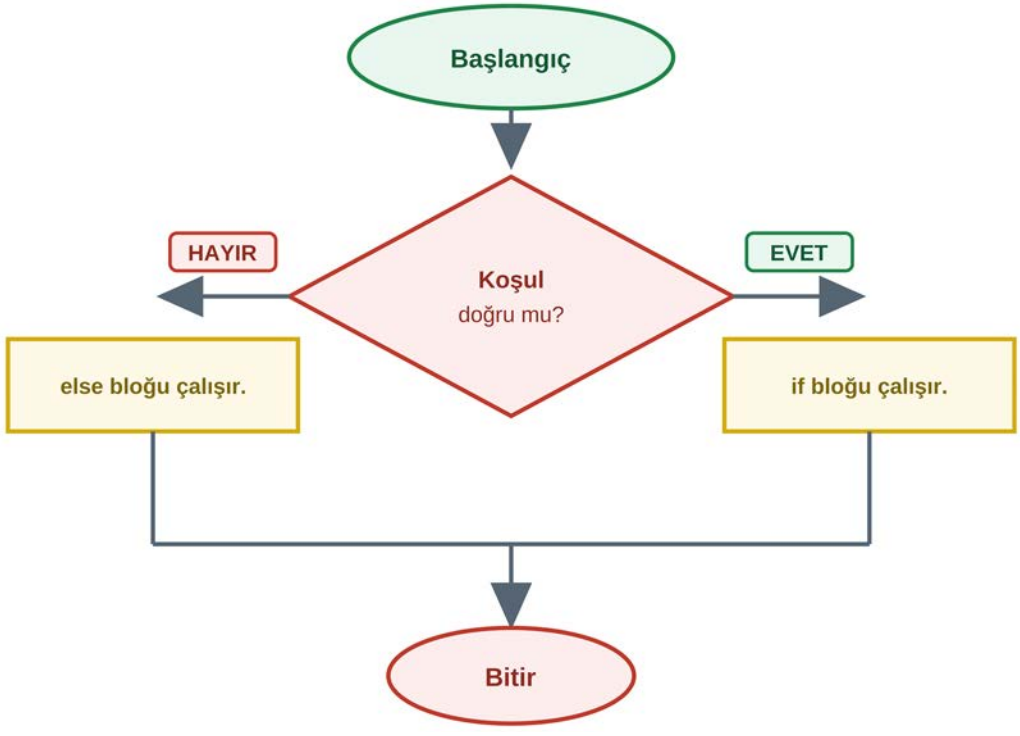
```
1   sicaklik = 32
2
3   if sicaklik > 30:
4       print("Hava çok sıcak.")
5   if sicaklik > 25:
6       print("Yazlık kıyafet giyebilirsiniz.")
7   if sicaklik < 10:
8       print("Hava soğuk.")
```

EKRAN ÇIKTISI

```
Hava çok sıcak.
Yazlık kıyafet giyebilirsiniz.
```

if-else Yapısı

Koşul doğru değilse alternatif bir işlem yapmak istendiğinde **else (aksi takdirde)** yapısı kullanılır. if-else yapısının nasıl çalıştığı Görsel 3.4'te gösterilmiştir.



Görsel 3.4: if-else karar yapısının akış şeması

22. ÖRNEK UYGULAMA

Amaç: if-else yapısı ile bir sayının pozitif mi negatif mi olduğunu belirlemek

```
1   sayi = int(input("Bir sayı giriniz: "))
2
3   if sayi >= 0:
4       print("Sayı pozitif (veya sıfır).")
5   else:
6       print("Sayı negatif.")
```

EKRAN ÇIKTISI

```
Bir sayı giriniz: -7
Sayı negatif.
```

if-elif-else Yapısı

İkiden fazla seçenek olduğunda ara koşullar için **elif (else if)** kullanılır. Program, koşulları yukarıdan aşağıya kontrol eder ve doğru olan ilk koşulun bloğunu çalıştırır.

23. ÖRNEK UYGULAMA

Amaç: if-elif-else yapısı ile not harfini belirlemek

```
1   puan = int(input("Puanınızı giriniz: "))
2
3   if puan >= 85:
4       print("Not: AA")
5   elif puan >= 70:
6       print("Not: BB")
7   elif puan >= 50:
8       print("Not: CC")
9   else:
10      print("Not: FF (Kaldınız)")
```

EKRAN ÇIKTISI

```
Puanınızı giriniz: 76
Not: BB
```

24. ÖRNEK UYGULAMA

Amaç: if-elif-else yapısıyla mevsim belirlemek

```
1   ay = int(input("Ay numarası (1–12): "))
2
3   if ay == 12 or ay == 1 or ay == 2:
4       print("Mevsim: Kış")
5   elif ay == 3 or ay == 4 or ay == 5:
6       print("Mevsim: İlkbahar")
7   elif ay == 6 or ay == 7 or ay == 8:
8       print("Mevsim: Yaz")
9   elif ay == 9 or ay == 10 or ay == 11:
10      print("Mevsim: Sonbahar")
11  else:
12      print("Geçersiz ay numarası!")
```

EKRAN ÇIKTISI

```
Ay numarası (1-12): 7
Mevsim: Yaz
```

SIRA SİZDE

Alışveriş uygulaması için indirim hesaplayan bir program yazınız. Kullanıcıdan alışveriş tutarını alıp ardından aşağıdaki işlemleri yapınız.

1. Tutar 500 TL ve üzeriyse %20 indirim uygulansın.
2. Tutar 250-500 TL arasındaysa %10 indirim uygulansın.
3. Tutar 250 TL'nin altındaysa indirim uygulanmasın.
4. if-elif-else yapısını kullanarak indirimli son tutarı ekrana yazdırınız.

8. UYGULAMA

Vücut Kitle İndeksi (VKİ) Hesaplayan Program Geliştiriyorum

Süre: 40 dakika

Amaç: Verilen fiziksel ölçülere (boy, kilo) göre vücut kitle indeksini hesaplayan ve sağlık durumunu koşul yapılarıyla gösteren uygulama geliştirmek

Etik ve Gizlilik Notu:

Uygulamada hiçbir öğrencinin gerçek boy ve kilo bilgisi kullanılmayacaktır. Uygulama yalnızca programlama mantığını öğrenmek amacıyla hazırlanmış sanal kişi verileri üzerinden yürütülecektir. Elde edilen sonuçlar tıbbi tanı veya kişisel sağlık değerlendirmesi olarak yorumlanmayacaktır.

Yönerge: Aşağıda sanal kişilere ait boy ve kilo bilgileri verilmiştir. Bu verileri kullanarak VKİ hesaplayan ve sonucu koşul yapılarıyla sınıflandıran bir program geliştiriniz.

Sanal Kişi	Boy(m)	Kilo(kg)
Kişi A	1,75	68
Kişi B	1,60	52
Kişi C	1,82	92

1. Adım: Editörünüzde vki_hesapla.py adında bir dosya oluşturunuz.

2. Adım: Programda sanal kişi için boy ve kilo değişkenlerini tanımlayınız.

3. Adım: Vücut kitle indeksini şu formülle hesaplayınız: VKİ = kilo / (boy * boy)

4. Adım: if-elif-else yapısıyla verilen tablodaki gibi sınıflandırma yapınız.

VKİ Aralığı	Durum
18,5'in altında	Zayıf
18,5 - 24,9 arası	Normal
25,0 - 29,9 arası	Fazla kilolu
30,0 ve üzeri	Obez

5. Adım: VKİ değerini ve VKİ aralığını ekrana açıklayıcı biçimde yazdırınız.

6. Adım: Programı verilen üç sanal kişi verisiyle test ederek sonuçları tablo halinde not ediniz.

7. Adım: Programınızı dijital deponuza yükleyiniz. README dosyasında uygulamanın gerçek kişisel veri kullanılmadan sanal verilerle yapıldığını belirtiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

İç İçe Karar Yapıları

Bir koşulun içinde başka koşullar da kontrol edilebilir. Bunlara **iç içe karar (nested) yapıları** denir. İç içe yapılarda dıştaki koşul doğru olduğunda içteki koşullar kontrol edilir.

25. ÖRNEK UYGULAMA

Amaç: İç içe karar yapısı kullanmak

```
1   yas = int(input("Yaşınız: "))
2   ehliyet = input("Ehliyetiniz var mı? (E/H): ")
3
4   if yas >= 18:
5       if ehliyet == "E":
6           print("Araç kullanabilirsiniz.")
7       else:
8           print("Önce ehliyet almalısınız.")
9   else:
10      print("18 yaşından küçüksünüz, araç kullanamazsınız.")
```

EKRAN ÇIKTISI
Yaşınız: 20
Ehliyetiniz var mı? (E/H): H
Önce ehliyet almalısınız.

Mantıksal Operatörlerle Karmaşık Koşullar

İç içe yapılar yerine, and ve or operatörleriyle aynı kontrol tek satırda yapılabilir. Bu, kodu daha kısa ve okunabilir kılar.

26. ÖRNEK UYGULAMA

Amaç: İç içe yapı yerine mantıksal operatör kullanmak

```
1   yas = 20
2   ehliyet = "E"
3
4   # İç içe yapı yerine and operatörü
5   if yas >= 18 and ehliyet == "E":
6       print("Araç kullanabilirsiniz.")
7   else:
8       print("Araç kullanamazsınız.")
```

EKRAN ÇIKTISI
Araç kullanabilirsiniz.

BİLGİ KUTUSU

İç içe karar yapıları ile mantıksal operatörlü koşullar çoğu zaman aynı işi yapar. Az sayıda koşul birlikte kontrol edilecekse and/or operatörleri daha sade bir çözüm sunar. Çok katmanlı ve farklı işlemler gerektiren durumlarda ise iç içe yapılar tercih edilir.

SIRA SİZDE

Eğlence parkına giriş için yaşın 12'den büyük ve boyun da 140 cm'den uzun olması şeklinde iki koşul vardır.

Kontrolü iki farklı biçimde yazınız.

1. iç içe if yapısı kullanarak
2. mantık operatörü kullanarak.
3. İki çözümü karşılaştırınız ve hangisinin daha okunabilir olduğunu açıklayınız.

9. UYGULAMA

İç İçe Koşul Yapılarıyla Ders Geçme Değerlendirmesi Geliştiriyorum

Süre: 40 dakika

Amaç: İç içe koşul yapılarıyla birden fazla durumu birlikte değerlendirmek

Yönerge: Bir öğrencinin ders notu, ödev teslimi ve devamsızlık durumu dikkate alınarak dersten geçip geçmediğini belirleyen uygulamayı geliştiriniz.

1. Adım: Editörünüzde ders_gecme.py adında bir dosya oluşturunuz.

2. Adım: Kullanıcıdan sınav notu (0-100), ödev teslim durumu (E/H) ve devamsızlık sayısı bilgilerini alınız.

3. Adım: Aşağıdaki kurallara göre değerlendirme yapınız.

- Devamsızlık 20'den fazlaysa öğrenci doğrudan kalır (diğer koşullara bakılmaz).
- Devamsızlık 20'nin altındaysa; sınav notu 50 ve üstüyse ödev teslim edilmişse öğrenci geçer.
- Sınav notu 50'nin altındaysa öğrenci kalır.
- Sınav notu 50 ve üstüyse fakat ödev teslim edilmemişse öğrenci bütünlemeye kalır.

4. Adım: İç içe if yapıları kullanarak yukarıdaki mantığı kodlayınız.

5. Adım: Programı (a) hep iyi, (b) devamsız, (c) sınav düşük, (d) ödev eksik şeklinde en az dört farklı senaryo ile test ediniz.

Senaryo (Durum)	Sınav Notu	Ödev Teslimi (E/H)	Devamsızlık	Beklenen Çıktı
a) Hep İyi	85	E	5	Öğrenci Geçti.
b) Devamsız	75	E	22	Doğrudan Kaldı.
c) Sınav Düşük	40	E	3	Öğrenci Kaldı.
d) Ödev Eksik	70	H	8	Bütünlemeye Kaldı.

6. Adım: Programınızı dijital deponuza yükleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Kontrol Listesi** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak kontrol listesine ulaşabilirsiniz.

Kontrol listesine ulaşmak için karekodu okutunuz.

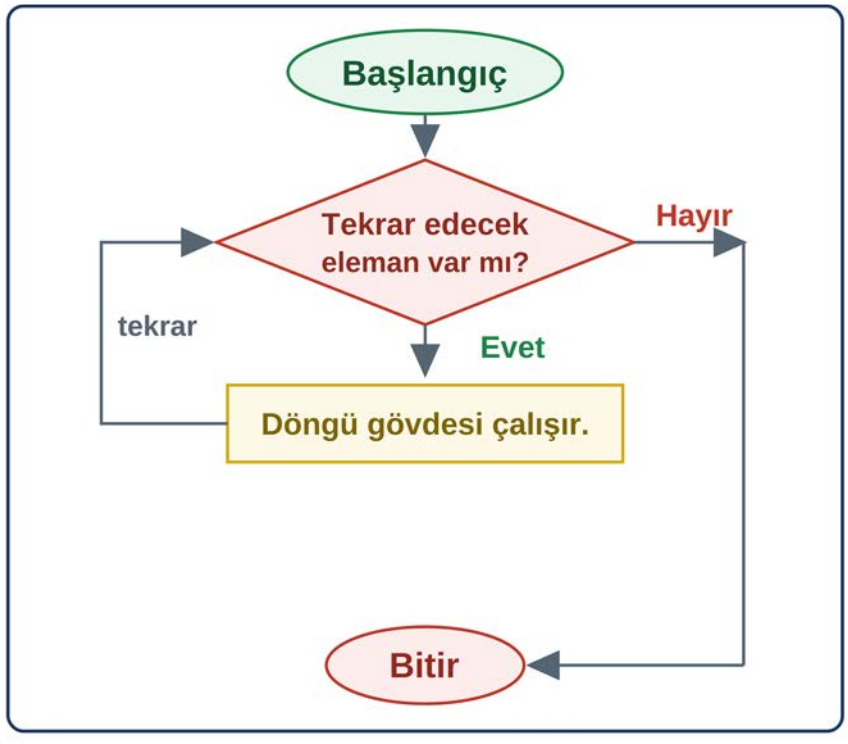
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

3.5. DÖNGÜ YAPILARIYLA TEKRARLI İŞLEMLER

Bir işi tek sefer yapmak yerine birçok kez yapmak istendiğinde döngü yapıları kullanılır. Bir öğretmen, yoklama sırasında her öğrenci için aynı işlemi tekrarlar. Programlamada da bu tür tekrarlı işler için döngüler kullanılır.

for Döngüsü

Belirli aralıkta veya koleksiyonun elemanları üzerinde tekrar yapmak için **for döngüsü** kullanılır. Döngünün çalışma mantığı Görsel 3.5'te gösterilmiştir.



Görsel 3.5: Döngünün çalışma mantığını gösteren akış şeması

for döngüsü genellikle range() fonksiyonu ile kullanılır. range() fonksiyonu belirlenen aralıkta değer üretmek için üç farklı şekilde kullanılır.

- range(5) ⟶ 0, 1, 2, 3, 4 (varsayılan başlangıç 0)
- range(2, 8) ⟶ 2, 3, 4, 5, 6, 7 (Başlangıç ve bitiş belirtilir.)
- range(0, 10, 2) ⟶ 0, 2, 4, 6, 8 (Üçüncü parametre adım miktarıdır.)

27. ÖRNEK UYGULAMA

Amaç: for döngüsü ile sayıları ekrana yazdırmak

```
1   # 1'den 5'e kadar sayıları yazdırma
2   for sayi in range(1, 6):
3       print("Sayı:", sayi)
```

EKRAN ÇIKTISI
Sayı: 1
Sayı: 2
Sayı: 3
Sayı: 4
Sayı: 5

28. ÖRNEK UYGULAMA

Amaç: for döngüsü ile toplam hesaplamak

```
1   toplam = 0
2
3   for sayi in range(1, 11):
4       toplam = toplam + sayi
5
6   print("1'den 10'a kadar sayıların toplamı:", toplam)
```

EKRAN ÇIKTISI
1'den 10'a kadar sayıların toplamı: 55

29. ÖRNEK UYGULAMA

Amaç: for döngüsü ile çarpım tablosu yazdırmak

```
1   sayi = 7
2
3   for i in range(1, 11):
4       print(sayi, "x", i, "=", sayi * i)
```

```
sayi = 7

for i in range(1, 11):
    print(sayi, "x", i, "=", sayi * i)
```

EKRAN ÇIKTISI
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70

SIRA SİZDE

for döngüsü kullanarak aşağıdaki işlemleri yapan programları yazınız.

1. 1'den 20'ye kadar olan çift sayıları ekrana yazdırınız.

2. Kullanıcıdan bir sayı alıp o sayının çarpım tablosunu (1'den 10'a kadar) yazdırınız.

while Döngüsü

Koşul doğru olduğu sürece tekrar etmek için **while döngüsü** kullanılır. for döngüsü genellikle belirli sayıda tekrar için while döngüsü ise tekrar sayısının önceden bilinmediği durumlar için uygundur.

30. ÖRNEK UYGULAMA

Amaç: while döngüsü ile geriye sayım yapmak

```
1   sayac = 5
2
3   while sayac > 0:
4       print("Kalan:", sayac)
5       sayac = sayac - 1
6
7   print("Süre doldu!")
```

EKRAN ÇIKTISI
Kalan: 5
Kalan: 4
Kalan: 3
Kalan: 2
Kalan: 1
Süre doldu!

while döngüsüyle koşul **(sayac > 0)** doğru olduğu sürece çalışılır. Her tekrarda sayac değeri bir azaltılır, sayac 0 olduğunda koşul yanlış olur ve döngü sona erer.

31. ÖRNEK UYGULAMA

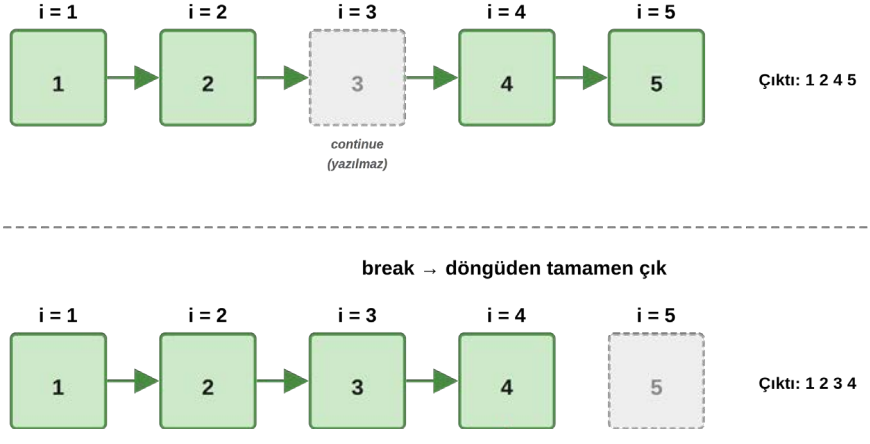
Amaç: while döngüsü ile toplam hesaplamak

```
1   sayi = 1
2   toplam = 0
3
4   while sayi <= 10:
5       toplam = toplam + sayi
6       sayi = sayi + 1
7
8   print("Toplam:", toplam)
```

EKRAN ÇIKTISI
Toplam: 55

Döngü Kontrol Deyimleri

Döngülerin akışını değiştirmek için **dur (break)** ve **devam et (continue)** adında iki özel deyim kullanılır. **break** döngüyü hemen sonlandırır. continue mevcut tekrarı atlar ve döngüye bir sonraki tekrar ile devam eder. Görsel 3.6'da iki deyimin döngüdeki görevi gösterilmiştir.



Görsel 3.6: break ve continue deyimlerinin çalışma mantığı

32. ÖRNEK UYGULAMA

Amaç: break deyimi ile döngüden çıkmak

```
1   for sayi in range(1, 100):
2       if sayi == 5:
3           print("5'e ulaşıldı, döngü sonlandırılıyor.")
4           break
5       print("Sayı:", sayi)
```

EKRAN ÇIKTISI
Sayı: 1
Sayı: 2
Sayı: 3
Sayı: 4
5'e ulaşıldı, döngü sonlandırılıyor.

33. ÖRNEK UYGULAMA

Amaç: continue deyimi ile bir tekrarı atlamak

continue deyimi kullanıldığında döngünün o anki tekrarı atlanır ama döngü tamamen bitirilmez.

```
1   for sayi in range(1, 8):
2       if sayi == 4:
3           continue        # 4'ü atla
4       print("Sayı:", sayi)
```

EKRAN ÇIKTISI
Sayı: 1
Sayı: 2
Sayı: 3
Sayı: 5
Sayı: 6
Sayı: 7

34. ÖRNEK UYGULAMA

Amaç: Döngü ile bir aralıkta sayı aramak

break deyimiyle aranan bir şey bulunduğunda döngü erken sonlandırılır.

```
# 1-50 arasında 7'ye tam bölünen ilk sayıyı bul
for sayi in range(1, 51):
    if sayi % 7 == 0:
        print("7'ye bölünen ilk sayı:", sayi)
        break
```

EKRAN ÇIKTISI
7'ye bölünen ilk sayı: 7

35. ÖRNEK UYGULAMA

Amaç: Şifre doğrulama sistemi geliştirmek

```
dogru_sifre = "1234"

while True:
    girilen = input("Şifrenizi giriniz: ")
    if girilen == dogru_sifre:
        print("Giriş başarılı!")
        break
    else:
        print("Hatalı şifre, tekrar deneyiniz.")
```

EKRAN ÇIKTISI
Şifrenizi giriniz: 0000 Hatalı şifre, tekrar deneyiniz. Şifrenizi giriniz: 1234 Giriş başarılı!

while True ifadesiyle sonsuz bir döngü oluşturulur. Döngü yalnızca doğru şifre girildiğinde **break** deyimi ile sonlandırılır.

SIRA SİZDE

Kullanıcıdan sürekli sayı alan bir program yazınız. Kullanıcı 0 girene kadar program sayı almaya devam etsin.

Program, girilen tüm sayıların toplamını ve adedini tutsun. Kullanıcı 0 girdiğinde while döngüsünü break komutuyla sonlandırarak toplam ve ortalamayı ekrana yazdırınız.

10. UYGULAMA

Döngülerle Sayı Dizileri ve Matematiksel Hesaplamalar Yapıyorum

Süre: 40 dakika

Amaç: for ve while döngülerini kullanarak matematiksel hesaplamalar yapmak

Yönerge: Üç ayrı dosya oluşturarak aşağıdaki problemleri çözünüz.

1. Adım: sayilar.py dosyasında, for döngüsü kullanarak 1'den 10'a kadar olan sayıları ekrana yazdırınız.

2. Adım: faktoriyel.py dosyasında kullanıcıdan bir sayı alıp n! = n × (n-1) × ... × 2 × 1 değerini hesaplayan programı yazınız.

BİLGİ NOTU

Fibonacci dizisi, ilk iki terimi 0 ve 1 olan sayı dizisidir. Dizideki her yeni terim, kendinden önce gelen iki terimin toplamı alınarak elde edilir.

Örneğin: 0, 1, 1, 2, 3, 5, 8, 13 . . .

3. Adım: fibonacci.py dosyasında kullanıcıdan kaç terim isteneceğini sorunuz. İlk iki terimi 0 ve 1 olarak başlatınız. Daha sonra her yeni terimi, kendinden önceki iki terimin toplamı olacak şekilde hesaplayarak Fibonacci dizisinin ilk N terimini ekrana yazdırınız.

4. Adım: Üç programı da çalıştırarak doğru çalıştıklarından emin olunuz.

5. Adım: Çalışmalarınızı dijital deponuza yükleyiniz. Her bir dosya için anlamlı commit mesajları yazınız.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

11. UYGULAMA

Sayı Tahmin Oyunu Geliştiriyorum

Süre: 40 dakika

Amaç: Kullanıcı girdisini koşul yapılarıyla değerlendirip while döngüsü ve break deyimleri ile sayı tahmin oyunu mantığı kurmak

Yönerge: Bilgisayarın bir sayı tuttuğu ve kullanıcının bu sayıyı tahmin etmeye çalıştığı bir oyun geliştiriniz.

1. Adım: Editörünüzde sayi_tahmin.py adında bir dosya oluşturunuz.

2. Adım: Programın başına gizli bir sayı belirleyiniz (42 gibi). İleride bu işlemi rastgele de yapabilirsiniz.

3. Adım: Sonsuz bir while döngüsü açınız. Döngü içinde kullanıcıdan tahmin alınız ve deneme sayısını artırınız.

4. Adım: Tahmini, gizli sayıyla karşılaştırınız. Doğru ise break ile çıkınız, yanlış ise büyük / küçük olduğunu söyleyiniz.

5. Adım: Programı çalıştırıp en az iki kez tahmin oyunu oynayınız.

6. Adım: Programınızı dijital deponuza yükleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.



Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



3.1. PROJE	
Konu	Kişisel Harçlık Yönetim Sistemi Geliştiriyorum
Kapsanan Konular	Algoritma geliştirme, bilgi girişi, bilgi yazdırma, değişken tanımlama, koşul yapıları, döngü yapıları
Süre	40+40+40 dakika
Amaç	Algoritma ve programlama yapılarını kapsayıcı bir projede kullanmak
Çalışma Biçimi	Bireysel

Proje Senaryosu

Lise birinci sınıf öğrencisisiniz. Anne ve babanız size her hafta belirli bir miktarda harçlık veriyor. Bu harçlığı bilinçli biçimde yönetmek istiyorsunuz. Bunun için harcamalarınızı kaydeden, toplam harcamanızı gösteren ve tasarruf hedefinize ulaşıp ulaşmadığınızı kontrol eden bir program geliştireceksiniz.

Yönerge: Aşağıdaki adımları takip ederek projeyi geliştiriniz.

1. Adım: Planlama (Algoritma): Projenizin tüm adımlarının algoritmasını yazınız ve akış şemasını çiziniz.

2. Adım: Geliştirme Ortamı: Editörünüzde harclik_yonetim.py dosyası oluşturunuz.

3. Adım: Kullanıcı Verileri (Veri Tipleri): Programa giriş yapacak öğrencinin adı (str), haftalık harçlığı (float) ve aylık tasarruf hedefi (float) bilgilerini alınız.

4. Adım: Menü (Koşullu İfadeler + Döngü): Bir while döngüsü içinde bir menü gösteriniz (1-Harcama Ekle, 2-Toplam Harcama, 3-Tasarruf Kontrolü, 4-Aylık Özet, 5-Çıkış). Kullanıcı çıkış yapana kadar program devam etsin.

5. Adım: Harcama Ekleme (Operatörler): Kullanıcı 1'i seçtiğinde harcama miktarı ve kategorisini (yemek/ulaşım/eğlence/kırtasiye/diğer) sorunuz.

6. Adım: Toplam Harcama Gösterme: Kullanıcı 2'yi seçtiğinde toplam harcamayı, kalan harçlığı ve harcama yüzdesini ekrana yazdırınız.

7. Adım: Tasarruf Kontrolü (İç İçe Koşullar): Kullanıcı 3'ü seçtiğinde kalan harçlık tasarruf hedefinden büyükse "Hedefinizi başardınız!" mesajı aksi takdirde "Hedefe ulaşamadınız." mesajı gösteriniz.

8. Adım: Aylık Özet (Döngü): Kullanıcı 4'ü seçtiğinde programa eklenmiş harcamaların tümünü for döngüsü sırasıyla yazdırınız.

9. Adım: Çıkış: Kullanıcı 5'i seçtiğinde "Görüşmek üzere, Ahmet!" gibi kişiselleştirilmiş bir mesajla döngüden break ile ayrılınız.

10. Adım: Test: (a) çok harcayan, (b) hedef tutturan, (c) çok tasarruflu olmak üzere en az üç farklı senaryoyla programı test ediniz.

Test Senaryosu	Haftalık Harçlık	Aylık Tasarruf Hedefi	Örnek Harcamalar	Beklenen Çıktı
Çok Harcayan	400 TL	300 TL	Yemek 350 TL, Ulaşım 100 TL	Hedefe ulaşamadınız.
Hedef Tutturan	500 TL	400 TL	Kırtasiye 200 TL, Diğer 100 TL	Hedefe ulaştınız.
Çok Tasarruflu	600 TL	200 TL	Eğlence 50 TL, Ulaşım 50 TL	Hedefe ulaştınız.

11. Adım: Dijital Depo: Projenizi dijital deponuza yükleyiniz. README.md dosyasında projenin amacını ve nasıl kullanılacağını yazınız.

Değerlendirme

a) Yapacağınız proje çalışması öğretmeniniz tarafından **Proje Derecelendirme Ölçeğinde** yer alan ölçütlere göre değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak derecelendirme ölçeğine ulaşabilirsiniz.

Proje derecelendirme ölçeğine ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

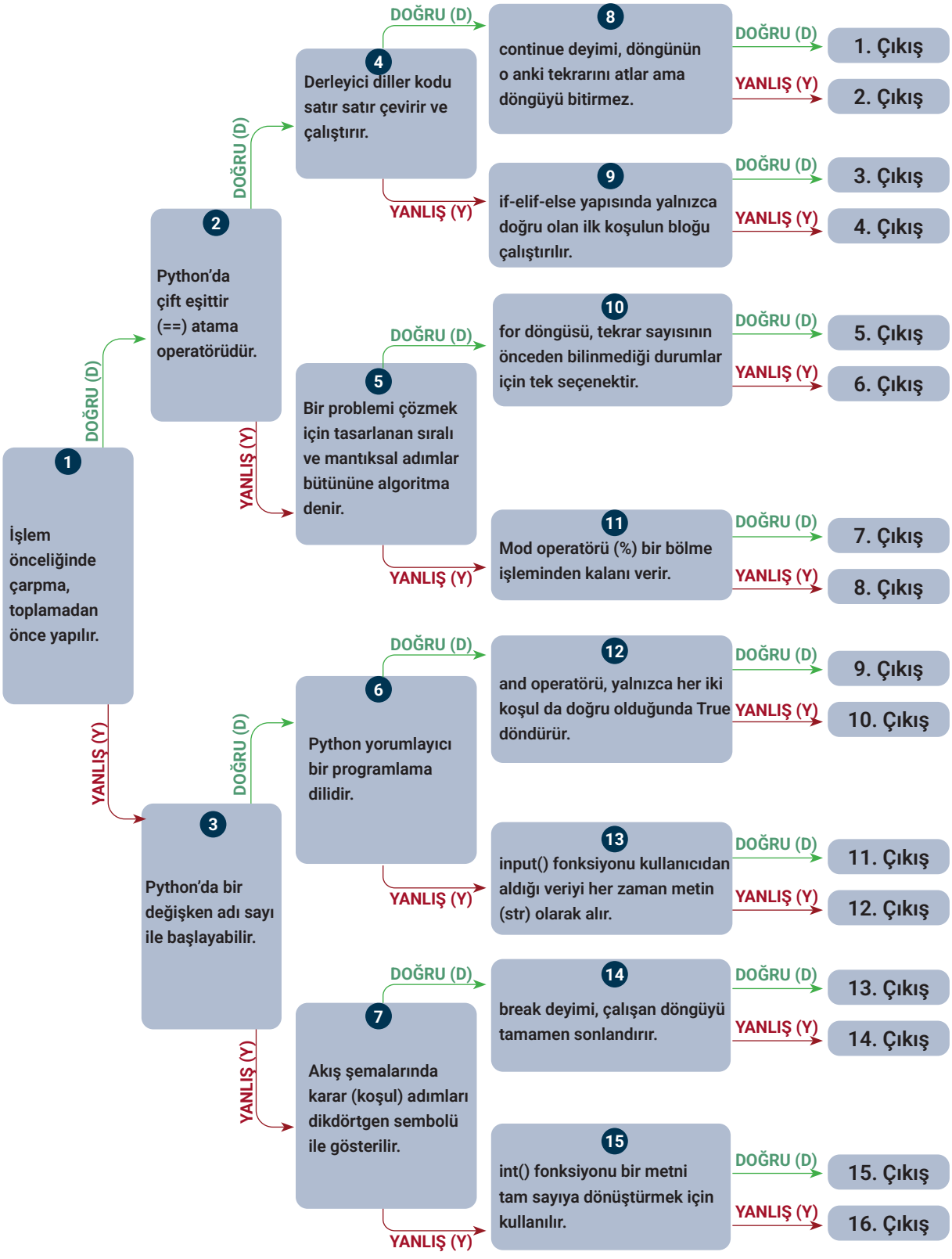
b) **Öz Değerlendirme Formu** proje geliştirme sürecinin sonunda kendinizi değerlendirmeniz amacıyla hazırlanmıştır. Cevaplarınızı tabloda ilgili kutucuğa yazınız. Verilen karekodu kullanarak öz değerlendirme formuna ulaşabilirsiniz.

Öz değerlendirme formuna ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

ÖLÇME VE DEĞERLENDİRME

A) Aşağıda Doğru / Yanlış önermeleri içeren tanılayıcı dallanmış ağaç verilmiştir. Verilen önermenin doğru olduğunu düşünüyorsanız (D), yanlış olduğunu düşünüyorsanız (Y) seçeneğini işaretleyiniz. İlk önermeden başlayarak yapacağınız seçimlerle ve oklar yönünde ilerleyerek bir çıkış noktası belirleyiniz.



B) Aşağıdaki cümlelerde boş bırakılan yerleri doğru kelimelerle doldurunuz.

1. Bir problemi çözmek için tasarlanan sıralı adımlar bütününe ..... denir.

2. Algoritmanın görsel gösterimine ..... denir.

3. Kodu satır satır çevirip çalıştıran dillere ..... diller denir.

4. Verilerin bellekte saklandığı, etiketlenmiş kutulara ..... denir.

5. Bir değişkenin tipini öğrenmek için ..... fonksiyonu kullanılır.

6. Kullanıcıdan alınan metni sayıya dönüştürme işlemine ..... denir.

7. Bir bölme işleminden kalanı veren operatöre ..... operatörü denir.

8. İki değerin eşit olup olmadığını kontrol eden operatör ..... işaretidir.

9. İkiden fazla seçenek olduğunda ara koşullar için ..... anahtar sözcüğü kullanılır.

10. Bir koşul doğru olduğu sürece tekrar eden döngü ..... döngüsüdür.

C) Sol sütundaki açıklamayı, sağ sütundaki doğru ifadelere ait harfle eşleştiriniz.

	Açıklama
( )	1. Kullanıcıdan veri almak için kullanılır.
( )	2. Sıralı ve mantıksal adımlar bütünü
( )	3. Bir koşula göre farklı işlem yapan yapı
( )	4. Verinin saklandığı etiketli kutu
( )	5. Tekrarlı işlemler için kullanılan yapı

İfade
a. Algoritma
b. Değişken
c. input()
ç. Karar yapısı
d. Döngü
e. if

Ç) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Akış şemalarında karar (koşul) adımları hangi sembolle gösterilir?

A) Daire
B) Dikdörtgen
C) Eşkenar Dörtgen
D) Oval
E) Paralelkenar

2. Aşağıdaki dillerden hangisi yorumlayıcı bir dildir?

A) C
B) C++
C) Go
D) Python
E) Rust

3. Aşağıdaki değişken adlarından hangisi geçerlidir?

A) 1sinif
B) dogum-tarihi
C) for
D) ogrenci_no
E) ders notu

4. input() fonksiyonu ile alınan veri varsayılan olarak hangi tiptedir?

A) bool
B) float
C) int
D) list
E) str

5. 10 % 3 işleminin sonucu nedir?

A) 0
B) 1
C) 3
D) 3.33
E) 30

6. 2 + 3 * 4 ifadesinin sonucu nedir?

A) 9
B) 11
C) 14
D) 20
E) 24

7.

```
puan = 65
if puan >= 70:
    print(“Geçti”)
else:
    print(“Kaldı”)
```

Yukarıdaki kodun çıktısı nedir?

A) Geçti
B) Hiçbir şey
C) 65
D) Hata
E) Kaldı

8. and operatörü ile ilgili aşağıdakilerden hangisi doğrudur?

A) Koşullardan biri doğruysa True döndürür.
B) Her zaman False döndürür.
C) Koşulun tersini döndürür.
D) Sadece sayılarla kullanılır.
E) Her iki koşul da doğruysa True döndürür.

9. range(2, 10, 2) ifadesi hangi sayıları üretir?

A) 2, 4, 6, 8
B) 2, 3, 4, ..., 10
C) 2, 4, 6, 8, 10
D) 0, 2, 4, 6, 8
E) 2, 10

10.

```
for i in range(1, 4):
    print(i)
```

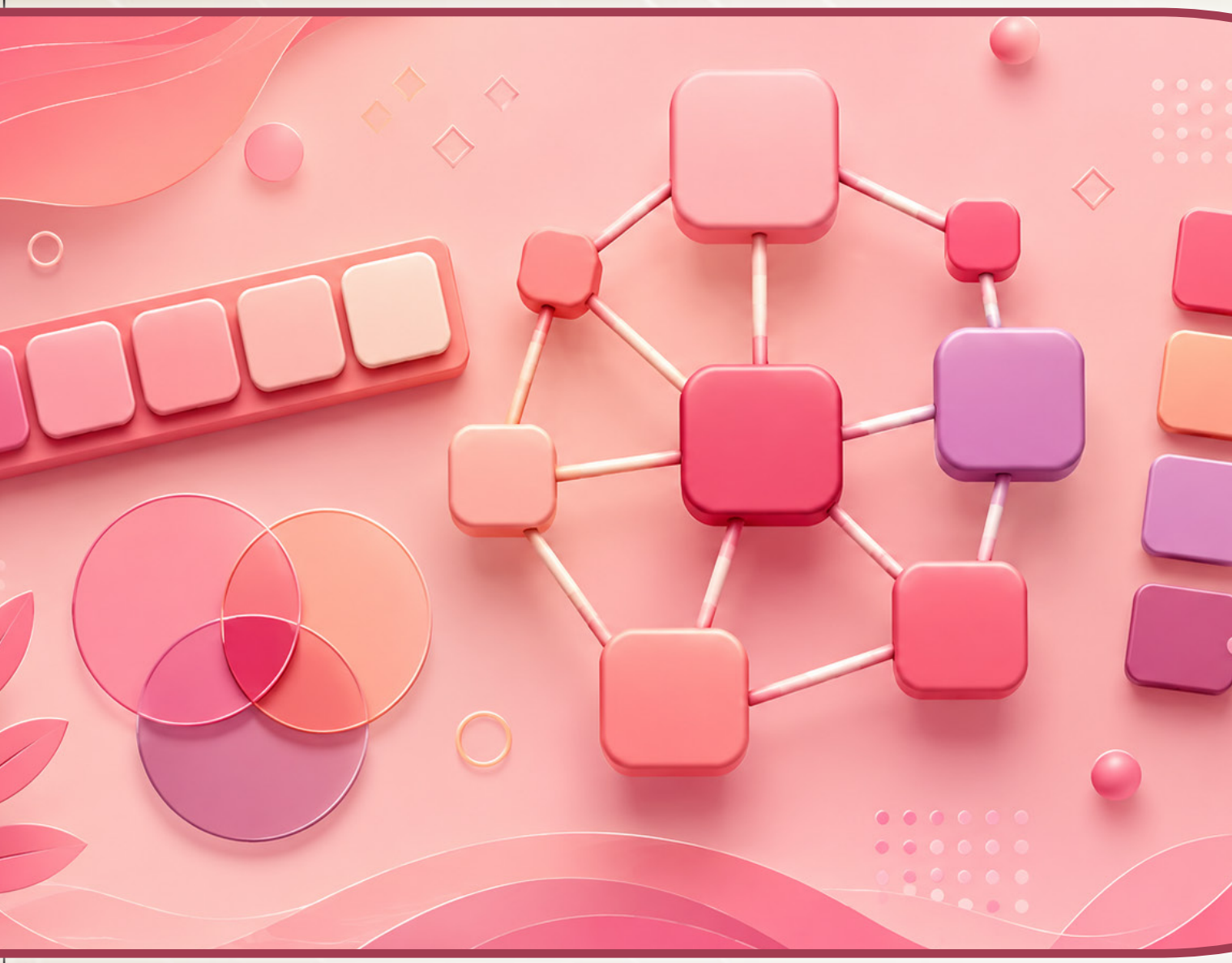
Yukarıdaki kodun çıktısı nedir?

A) 1 2 3
B) 0 1 2 3
C) 1 2 3 4
D) 0 1 2
E) 1 4

D) Bu öğrenme biriminde edindiklerinizi değerlendirmek için aşağıdaki çıkış kartını doldurunuz.

Üç Yaz	Programlamaya Giriş öğrenme biriminde öğrendiğiniz üç bilgiyi yazınız.
1	
2	
3	
İki Sor	Programlama ile ilgili merak ettiğiniz iki soruyu yazınız.
1	
2	
Bir Paylaş	Bu öğrenme biriminde geliştirdiğiniz uygulamalardan birini sınıf arkadaşlarınızla paylaşmak isteseniz hangisini seçerdiniz? Neden?





KONULAR

4.1. LİSTE VERİ YAPILARI

4.2. DEMET VERİ YAPISI

4.3. KÜME VERİ YAPILARI

4.4. SÖZLÜK VERİ YAPISI

ANAHTAR KAVRAMLAR

- anahtar-değer
- demet
- dilimleme
- indeks
- koleksiyon
- küme
- liste
- sözlük
- veri yapısı



4. ÖĞRENME BİRİMİ

VERİ KOLEKSİYONLARI

BEKLENTİLER

Bu öğrenme biriminde sizden

- liste veri yapılarını kullanmanız
- demet veri yapısını kullanmanız
- küme veri yapılarını kullanmanız
- sözlük veri yapısını kullanmanız

beklenmektedir.

HAZIRLIK ÇALIŞMALARI

1. Aşağıda günlük hayattan bir örnek verilmiştir.

"Her sabah mahalledeki manav tezgâhında bir kargaşa yaşanır. Domatesler kasaların üstünde, salatalıklar yerde, biberler farklı kutularda olduğundan tezgâh çok dağınık görünür. Bu yüzden müşteriler istedikleri ürünü ararken fazla zaman harcamak zorunda kalırlar. Dükkâna girenler manavın telaşla "Bu sabah hangi üründen ne kadar kaldığını ve ürünleri hangi fiyattan sattığımı hatırlamıyorum!" diye yakındığını duyarlar.

Karşıdaki manavda ürünler çeşit ve fiyatlarına göre raflara düzenli bir şekilde yerleştirilmiş, ürünlerin üzerine fiyat etiketleri de konulmuştur. Her ürünün barkodu olduğu için hangi kategoride bulunduğu ve fiyatı, sistemde kayıtlıdır. Böylece müşteri istediği ürünü hızlıca bulur, ödeme noktasındaki görevli fiyatı görerek işlem yapar ve gün sonunda satışlar kolayca raporlanır."

Verilen metne göre aşağıdaki soruları cevaplayınız.

a) İki manav arasındaki temel fark nedir?

b) Günlük hayatınızda hangi bilgileri (örneğin müzik listeniz, telefon rehberiniz, ders programınız) düzenliyorsunuz?

2. Bir e-ticaret sitesindeki ürün bilgileri sizce nasıl saklanır? Bu bilgileri düzenli tutmanın ne faydaları olabilir?

BAŞLARKEN

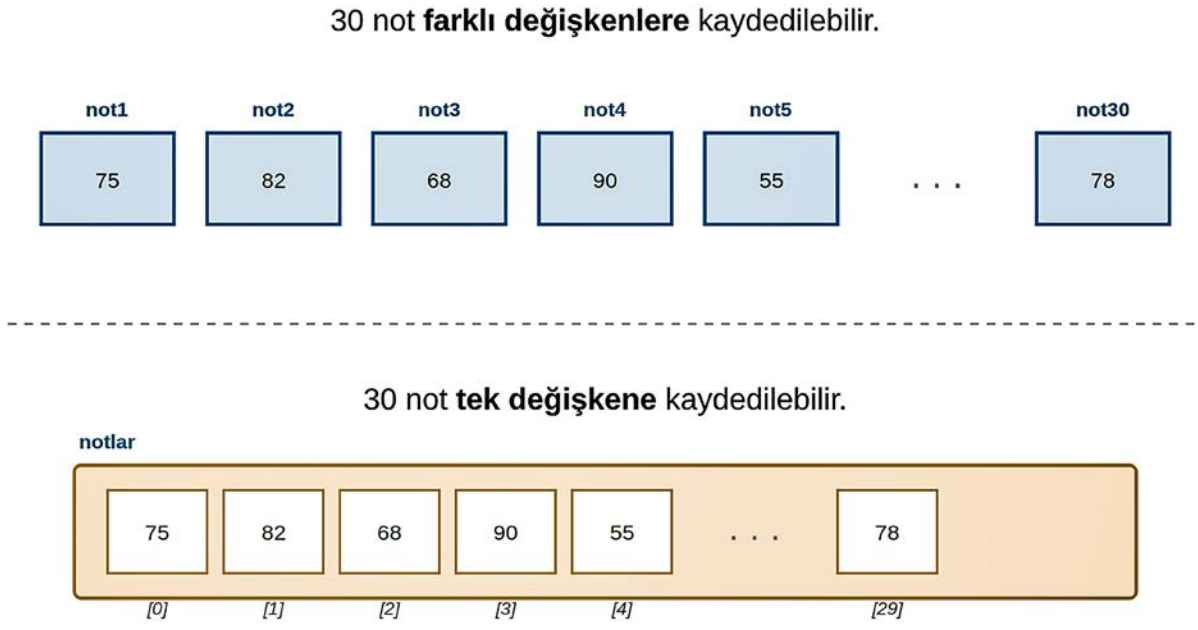
Bilginin düzenli tutulması ona ulaşmayı ve kullanmayı kolaylaştırır. Program geliştirirken her veri için farklı değişken tanımlamak yetersiz kalır. Örneğin 30 öğrencinin notunu 30 ayrı değişkende tutmak zahmetli ve yönetilmesi güçtür.

Sorunun çözümü **veri koleksiyonlarıdır.** Veri koleksiyonu, birbiriyle ilişkili pek çok verinin tek bir isim altında düzenli biçimde saklanmasını sağlayan yapıdır. Bir koleksiyon sayesinde 30 öğrencinin notu da 500 öğrencinin notu da tek bir değişkende tutulabilir.

Python'da **liste** (sıralı veri), **demet** (değiştirilemez veri), **küme** (benzersiz veri) ve **sözlük** (anahtar-değer verisi) olmak üzere dört temel veri koleksiyonu bulunur. Bu yapılardan her birinin kendine özgü kullanım alanları vardır. Doğru veri yapısının seçilmesi kodun hızı ve okunabilirliği üzerinde doğrudan etkilidir.

4.1. LİSTE VERİ YAPILARI

Veri koleksiyonları arasında en sık kullanılan yapı listedir. Liste, birden fazla veriyi tek bir değişken altında sıralı biçimde saklamaya yarayan bir veri yapısıdır. Bir sınıftaki tüm öğrencilerin notları, bir spor takımındaki oyuncular veya kütüphanedeki kitaplar tek bir listede tutulabilir. Görsel 4.1'de değişken içinde tek bir veri saklama ile liste kullanarak birden fazla veriyi saklama arasındaki fark gösterilmiştir.



Görsel 4.1: Değişken ile liste veri yapısı arasındaki fark

Liste Oluşturma ve Elemanlara Erişim

Python'da bir liste, köşeli parantez içine virgülle ayrılmış elemanlar yazılarak oluşturulur. Liste; sayı, metin gibi farklı türde verileri bir arada tutabilir.

1. ÖRNEK UYGULAMA

Amaç: Farklı türde veri içeren listeler oluşturmak

```
1   # Sayısal liste
2   notlar = [75, 82, 68, 90, 55]
3
4   # Metinsel liste
5   ogrenciler = ["Ahmet", "Ayşe", "Mehmet", "Zeynep"]
6
7   # Karışık liste (farklı türlerde)
8   karisik = ["Ali", 15, 1.72, True]
9
10  # Boş liste
11  liste = []
```

Bu listeler metin ve sayısal değerler içermektedir.

2. ÖRNEK UYGULAMA

Amaç: Bir sınıfın notlarını listede toplamak

Listeler, çok sayıda benzer veriyi tek bir değişkende toplamak için idealdir.

```
1  notlar = [70, 85, 90, 65, 100, 55, 80]
2
3  print("Toplam not sayısı:", len(notlar))
4  print("İlk öğrencinin notu:", notlar[0])
5  print("Son öğrencinin notu:", notlar[-1])
6  print("Tüm notlar:", notlar)
```

EKRAN ÇIKTISI

Toplam not sayısı: 7

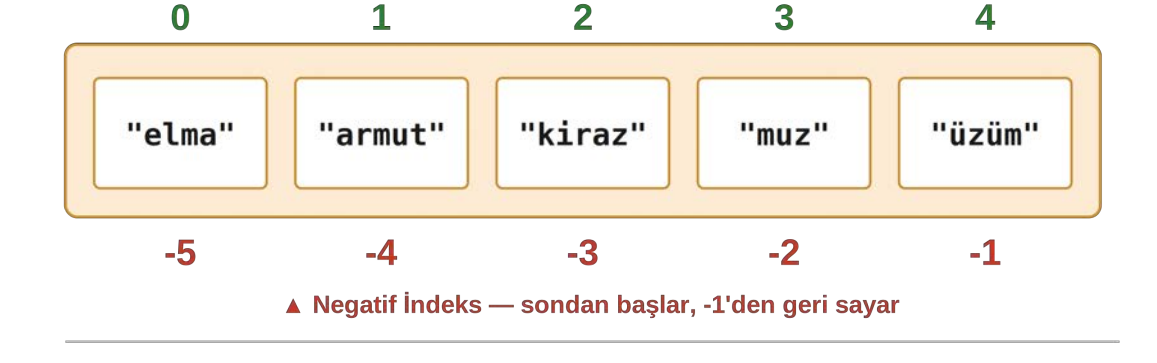
İlk öğrencinin notu: 70

Son öğrencinin notu: 80

Tüm notlar: [70, 85, 90, 65, 100, 55, 80]

Yedi öğrencinin notu tek bir liste değişkeninde toplanmıştır. Öğrenci sayısı 7 yerine 700 olsaydı yine tek bir liste değişkeni yeterli olurdu. len() fonksiyonu, bir listedeki eleman sayısını döndürür. Burada notlar isimli listenin 7 elemanı olduğu için len() fonksiyonu 7 sayısını geri döndürür.

Listenin elemanlarına **indeks** (sıra numarası) kullanılarak erişilir. İndeksleme 0'dan başlar. Listenin ilk elemanına ulaşmak için liste[0] ifadesi kullanılır. Python'da listelere sondan (geriye doğru) erişmek için negatif indeks numaraları da kullanılır. Listenin en son elemanının indeksi her zaman -1'dir ve sola doğru -2, -3 şeklinde azalır.İndeksleme mantığı Görsel 4.2 de gösterilmiştir.



Görsel 4.2: Liste elemanlarına indeks ile erişim

3. ÖRNEK UYGULAMA

Amaç: Liste elemanlarına pozitif ve negatif indekslerle erişmek

```
1   notlar = [75, 82, 68, 90, 55]
2
3   print(notlar[0])        # 75   (ilk eleman)
4   print(notlar[2])        # 68   (üçüncü eleman)
5   print(notlar[-1])       # 55   (sondan birinci)
6   print(notlar[-2])       # 90   (sondan ikinci)
7   # Hata: indeks aralık dışıysa IndexError
8   print(notlar[10])     # HATA
```

EKRAN ÇIKTISI

```
75
68
55
90
Traceback (most recent call last):
  File "C:\kodlar\4\4_2.py", line 8, in <module>
    print(notlar[10])     # HATA
          ~~~~~~^^^^
IndexError: list index out of range
```

DİKKAT

Bir listenin son elemanının indeksi, eleman sayısının bir eksiğidir. Beş elemanlı bir listenin geçerli indeksleri 0, 1, 2, 3 ve 4'tür. Listede olmayan bir indekse [örneğin notlar(10)] erişmeye çalışmak IndexError hatasına yol açar.

Listeler değiştirilebilir yapılardır, bir elemanın değeri indeksi kullanılarak değiştirilebilir.

4. ÖRNEK UYGULAMA

Amaç: Bir liste elemanının değerini değiştirmek

```
1  renkler=["kırmızı","yeşil","mavi"]
2  print("Değişiklik öncesi:",renkler)
3
4  renkler[1]="sarı"
5  print("Değişiklik sonrası:",renkler)
```

EKRAN ÇIKTISI
Değişiklik öncesi: [‘kırmızı’, ‘yeşil’, ‘mavi’]
Değişiklik sonrası: [‘kırmızı’, ‘sarı’, ‘mavi’]

5. ÖRNEK UYGULAMA

Amaç: Listede eleman olup olmadığını kontrol etmek

Bir elemanın listede olup olmadığı in operatörü ile kontrol edilir.

```
1  meyveler = ["elma", "armut", "kiraz"]
2
3  print("armut listede mi?", "armut" in meyveler)
4  print("muz listede mi?", "muz" in meyveler)
5
6  if "elma" in meyveler:
7      print("Elma stokumuzda var.")
```

EKRAN ÇIKTISI
armut listede mi? True
muz listede mi? False
Elma stokumuzda var.

in operatörü, bir elemanın listede bulunup bulunmadığını **True** veya **False** olarak döndürür. Bu özellik, koşul ifadeleriyle birlikte sıkça kullanılır.

SIRA SİZDE

1. Haftanın yedi gününü içeren gunler adında bir liste oluşturunuz. Bu liste üzerinde aşağıdaki işlemleri yapınız.

a) Listenin ilk ve son elemanını ekrana yazdırınız.
b) Listenin eleman sayısını len() ile bulunuz.
c) Negatif indeks kullanarak sondan üçüncü günü yazdırınız.
d) Listenin 2 indeks numaralı (baştan üçüncü sıradaki) elemanını "Çarşamba" olarak güncelleyip listeyi tekrar ekrana yazdırınız.

2. Bir basketbol takımındaki beş oyuncunun adını içeren bir liste oluşturunuz ve aşağıdaki işlemleri yapınız.

a) İkinci ve dördüncü oyuncuyu indeks kullanarak yazdırınız.
b) Takımdaki oyuncu sayısını len() fonksiyonunu kullanarak bulunuz.
c) Bir oyuncunun takımda olup olmadığını in operatörü ile kontrol ediniz.
d) Birinci oyuncuyu başka bir oyuncuyla değiştiriniz.

Liste Dilimleme

Bir listenin yalnızca belirli bir aralığını almak için **dilimleme (slicing)** kullanılır. Dilimlemenin genel söz dizimi şu şekildedir:

```
liste[başlangıç:bitiş:adım]
```

Dilimlemede başlangıç indeksi dâhil, bitiş indeksi dâhil değildir. Adım değeri ise kaçar kaçar ilerleneceğini belirtir.

6. ÖRNEK UYGULAMA

Amaç: Bir listenin belirli aralıklarını dilimlemek

```
1   notlar = [75, 82, 68, 90, 55, 73, 88]
2
3   print(notlar[1:4])      # [82, 68, 90]   (1, 2 ve 3. indeksler)
4   print(notlar[:3])       # [75, 82, 68]   (baştan 3 eleman)
5   print(notlar[3:])       # [90, 55, 73, 88] (3. indeksten sona)
6   print(notlar[::2])      # [75, 68, 55, 88] (ikişer atlayarak)
7   print(notlar[::-1])     # [88, 73, 55, 90, 68, 82, 75] (tersten)
```

EKRAN ÇIKTISI
82, 68, 90]
[75, 82, 68]
[90, 55, 73, 88]
[75, 68, 55, 88]
[88, 73, 55, 90, 68, 82, 75]

7. ÖRNEK UYGULAMA

Amaç: Dilimleme ile bir listeyi ters çevirmek

Adım değeri negatif verildiğinde liste tersten alınır. Bu yöntem, bir listeyi ters çevirmenin pratik bir yoludur.

```
1  harfler=["A","B","C","D","E"]
2  print(harfler[::-1])
```

EKRAN ÇIKTISI
['E', 'D', 'C', 'B', 'A']

8. ÖRNEK UYGULAMA

Amaç: Dilimleme ile bir listenin parçalarını işlemek

Dilimleme, bir listeyi mantıklı parçalara ayırmak için kullanılabilir. Aşağıdaki örnekte bir sınıf iki gruba ayrılmıştır.

```
1  sinif = ["Ali", "Ayşe", "Can", "Deniz", "Ece", "Fatma"]
2
3  birinci_grup = sinif[:3]
4  ikinci_grup = sinif[3:]
5
6  print("Birinci grup:", birinci_grup)
7  print("İkinci grup:", ikinci_grup)
```

EKRAN ÇIKTISI
Birinci grup: ['Ali', 'Ayşe', 'Can']
İkinci grup: ['Deniz', 'Ece', 'Fatma']

Dilimleme ile bir liste, herhangi bir noktadan ikiye bölünebilir. Bu yöntem, verileri gruplara ayırmak gerektiğinde sıkça kullanılır.

SIRA SİZDE

1'den 10'a kadar olan sayıları içeren bir liste oluşturunuz. Dilimleme ile aşağıdaki işlemleri yapınız.

1. İlk beş sayıyı ayrı bir listeye alıp yazdırınız.

2. Son üç sayıyı yazdırınız.

3. Yalnızca çift indeksli elemanları (adım= 2 ile) yazdırınız.

4. Listeyi dilimleme yöntemiyle ters çevirip yazdırınız.

Liste Metotları

Listeler üzerinde sık yapılan işlemler için Python'da hazır **metotlar** bulunur. Metot, bir veri yapısına ait özel bir fonksiyondur ve **liste.metot()** biçiminde çağrılır. En çok kullanılan liste metotları Tablo 4.1'de gösterilmiştir.

Tablo 4.1: Sık Kullanılan Liste Metotları

Metot	Görevi	Örnek Kullanım
append(x)	Listenin sonuna x elemanını ekler.	liste.append(5)
insert(i, x)	i indeksine x elemanını ekler.	liste.insert(0, 9)
remove(x)	x değerine sahip ilk elemanı siler.	liste.remove(5)
pop(i)	i indeksindeki elemanı siler ve döndürür.	liste.pop(2)
sort()	Listeyi küçükten büyüğe sıralar.	liste.sort()
reverse()	Listeyi ters çevirir.	liste.reverse()
index(x)	x elemanının indeksini bulur.	liste.index(9)
count(x)	x elemanının kaç kez geçtiğini sayar.	liste.count(9)

9. ÖRNEK UYGULAMA

Amaç: Listeye eleman ekleme ve silme metotlarını kullanmak

```
1  sepet = ["elma", "armut"]
2  print("Başlangıç:", sepet)
3
4  sepet.append("kiraz")        # sona ekler
5  print("append sonrası:", sepet)
6
7  sepet.insert(0, "muz")       # başa ekler
8  print("insert sonrası:", sepet)
9
10 sepet.remove("armut")        # armut'u siler
11 print("remove sonrası:", sepet)
```

EKRAN ÇIKTISI

```
Başlangıç: ['elma', 'armut']
append sonrası: ['elma', 'armut', 'kiraz']
insert sonrası: ['muz', 'elma', 'armut', 'kiraz']
remove sonrası: ['muz', 'elma', 'kiraz']
```

10. ÖRNEK UYGULAMA

Amaç: Listeyi sıralama ve ters çevirme metotlarını kullanmak

```
1  sayilar = [40, 10, 90, 20, 70]
2
3  sayilar.sort()
4  print("Sıralanmış:", sayilar)
5
6  sayilar.reverse()
7  print("Ters çevrilmiş:", sayilar)
8
9  print("90 sayısının indeksi:", sayilar.index(90))
```

EKRAN ÇIKTISI
Sıralanmış: [10, 20, 40, 70, 90]
Ters çevrilmiş: [90, 70, 40, 20, 10]
90 sayısının indeksi: 0

11. ÖRNEK UYGULAMA

Amaç: Metin listesini alfabetik sıralamak

sort() metodu, metinlerden oluşan listeleri alfabetik sıraya dizer.

```
1  ogrenciler = ["Zeynep", "Ali", "Mehmet", "Buse", "Can"]
2
3  ogrenciler.sort()
4  print("Alfabetik sıralı:", ogrenciler)
5
6  # Tersten sıralama
7  ogrenciler.sort(reverse=True)
8  print("Ters alfabetik:", ogrenciler)
```

EKRAN ÇIKTISI
Alfabetik sıralı: ['Ali', 'Buse', 'Can', 'Mehmet', 'Zeynep']
Ters alfabetik: ['Zeynep', 'Mehmet', 'Can', 'Buse', 'Ali']

sort() metodu; sayıları küçükten büyüğe, metinleri ise alfabetik sıraya dizer. **reverse=True** parametresi verilirse sıralama tersten yapılır.

BİLGİ KUTUSU

Liste metotları iki gruba ayrılır. append(), sort(), remove() gibi metotlar listeyi doğrudan değiştirir ve genellikle bir değer döndürmez. index(), count() gibi metotlar ise listeyi değiştirmez yalnızca bir bilgi (sonuç) döndürür.

12. ÖRNEK UYGULAMA

Amaç: pop() ve count() metotlarını kullanmak

```
1  puanlar = [10, 25, 10, 40, 10, 60]
2
3  # pop() ile eleman silme ve döndürme
4  silinen = puanlar.pop(1)
5  print("Silinen eleman:", silinen)
6  print("Güncel liste:", puanlar)
7
8  # count() ile bir elemanın kaç kez geçtiğini sayma
9  print("10 sayısı kaç kez var:", puanlar.count(10))
```

EKRAN ÇIKTISI
Silinen eleman: 25
Güncel liste: [10, 10, 40, 10, 60]
10 sayısı kaç kez var: 3

pop() metodu, sildiği elemanı aynı zamanda döndürür ve bu yüzden silinen değer bir değişkene atanabilir. **count() metodu** ise bir elemanın listede kaç kez geçtiğini sayar.

SIRA SİZDE

Boş bir alisveris_listesi oluşturunuz. Liste metotlarını kullanarak aşağıdaki işlemleri sırayla yapınız.

1. append() ile listeye üç ürün ekleyiniz.

2. insert() ile listenin başına bir ürün ekleyiniz.

3. remove() ile bir ürünü listeden çıkarınız.

4. sort() ile listeyi alfabetik sıraya diziniz.

Listede Döngüyle Gezinme

for döngüsü, listelerle birlikte kullanıldığında çok güçlü bir araçtır. Bir listenin tüm elemanları üzerinde tek tek işlem yapmak için for döngüsü kullanılır.

13. ÖRNEK UYGULAMA

Amaç: Bir listenin elemanlarını for döngüsü ile işlemek

```
1  notlar = [70, 85, 90, 65, 100]
2  toplam = 0
3
4  for not_degeri in notlar:
5      toplam = toplam + not_degeri
6
7  print("Notların toplamı:", toplam)
8  print("Notların ortalaması:", toplam / len(notlar))
```

EKRAN ÇIKTISI
Notların toplamı: 410
Notların ortalaması: 82.0

Döngü, listenin her elemanını sırayla not_degeri değişkenine atar ve gövdedeki işlemi her eleman için tekrarlar. Böylece listenin tüm elemanları toplanmış olur.

14. ÖRNEK UYGULAMA

Amaç: Döngü ile bir listede koşullu işlem yapmak

Döngü içinde koşul kullanarak listenin yalnızca belirli elemanları üzerinde işlem yapılabilir.

```
1   notlar = [75, 82, 68, 90, 55]
2   # Tüm notları yazdır
3   for not_ in notlar:
4       print(not_)
5   # Ortalama hesabı
6   toplam = 0
7   for not_ in notlar:
8       toplam = toplam + not_
9   ortalama = toplam / len(notlar)
10  print("Ortalama:", ortalama)
```

EKRAN ÇIKTISI
75
82
68
90
55
Ortalama: 74.0

15. ÖRNEK UYGULAMA

Amaç: Döngü ile bir listede arama yapmak

Bir listede belirli bir elemanın olup olmadığı döngü ile aranabilir. Aşağıdaki örnek, bir öğrencinin listede olup olmadığını kontrol eder.

```
1  ogrenciler = ["Ayşe", "Mehmet", "Zeynep", "Can", "Elif"]
2  aranan = "Zeynep"
3  bulundu = False
4
5  for ogrenci in ogrenciler:
6      if ogrenci == aranan:
7          bulundu = True
8
9  if bulundu:
10     print(aranan, "listede bulundu.")
11 else:
12     print(aranan, "listede yok.")
```

EKRAN ÇIKTISI

```
Zeynep listede bulundu.
```

Örnekteki döngü, listenin her elemanını aranan değerle karşılaştırır. Aynı sonuç in operatörü ile tek satırda da elde edilebilir ancak döngü ile yapılan arama, çalışma mantığını kavramak için önemlidir.

BİLGİ KUTUSU

Listenin indeksine ve elemanına aynı anda erişmek için enumerate() fonksiyonu kullanılabilir; **for indeks, eleman in enumerate(liste)** yazımı, her dönüşte sıra numarasının ve o sıradaki elemanı verir.

16. ÖRNEK UYGULAMA

Amaç: enumerate() ile listeyi sıra numarasıyla yazdırmak

```
1  ogrenciler = ["Ayşe", "Mehmet", "Zeynep", "Can"]
2
3  for sira, ad in enumerate(ogrenciler):
4      print(sira, "-", ad)
```

EKRAN ÇIKTISI
0 - Ayşe
1 - Mehmet
2 - Zeynep
3 - Can

enumerate() fonksiyonu, bir listenin elemanlarını sıra numarasıyla birlikte vermesi gereken durumlarda (örneğin numaralı bir liste yazdırılırken) çok kullanışlıdır.

17. ÖRNEK UYGULAMA

Amaç: Döngü ile yeni bir liste oluşturmak

Bir döngü kullanılarak var olan bir listeden yeni bir liste üretilebilir. Aşağıdaki örnek fiyatlara zam uygulanarak yeni bir liste oluşturur.

```
1  eski_fiyatlar = [100, 250, 80, 300]
2  yeni_fiyatlar = []
3
4  for fiyat in eski_fiyatlar:
5      zamli = fiyat * 1.20
6      yeni_fiyatlar.append(zamli)
7
8  print("Eski fiyatlar:", eski_fiyatlar)
9  print("Yeni fiyatlar:", yeni_fiyatlar)
```

EKRAN ÇIKTISI
Eski fiyatlar: [100, 250, 80, 300]
Yeni fiyatlar: [120.0, 300.0, 96.0, 360.0]

Örnekte boş bir liste oluşturulur ve döngü içinde append() metoduyla her hesaplanan değer bu listeye eklenir. Programlamada bu kalıp çok sık kullanılır.

18. ÖRNEK UYGULAMA

Amaç: Liste elemanlarını sayısal ölçütlere göre ayırmak

Bir döngü ile bir listenin elemanları farklı ölçütlere göre ayrı listelere ayrılabilir.

```
1  sayilar = [12, 7, 24, 9, 30, 15, 8]
2  ciftler = []
3  tekler = []
4
5  for s in sayilar:
6      if s % 2 == 0:
7          ciftler.append(s)
8      else:
9          tekler.append(s)
10
11 print("Çift sayılar:", ciftler)
12 print("Tek sayılar:", tekler)
```

EKRAN ÇIKTISI
Çift sayılar: [12, 24, 30, 8]
Tek sayılar: [7, 9, 15]

Döngü, her sayıyı kontrol ederek onu uygun listeye ekler. Bu yöntem, bir veri kümesini belirli ölçütlere göre gruplara ayrılması gerektiğinde kullanılır.

SIRA SİZDE

Bir mağazadaki ürün fiyatlarını içeren fiyatlar = [25, 80, 150, 45, 200, 90] listesini oluşturunuz.

for döngüsü kullanarak aşağıdaki işlemleri yapınız.

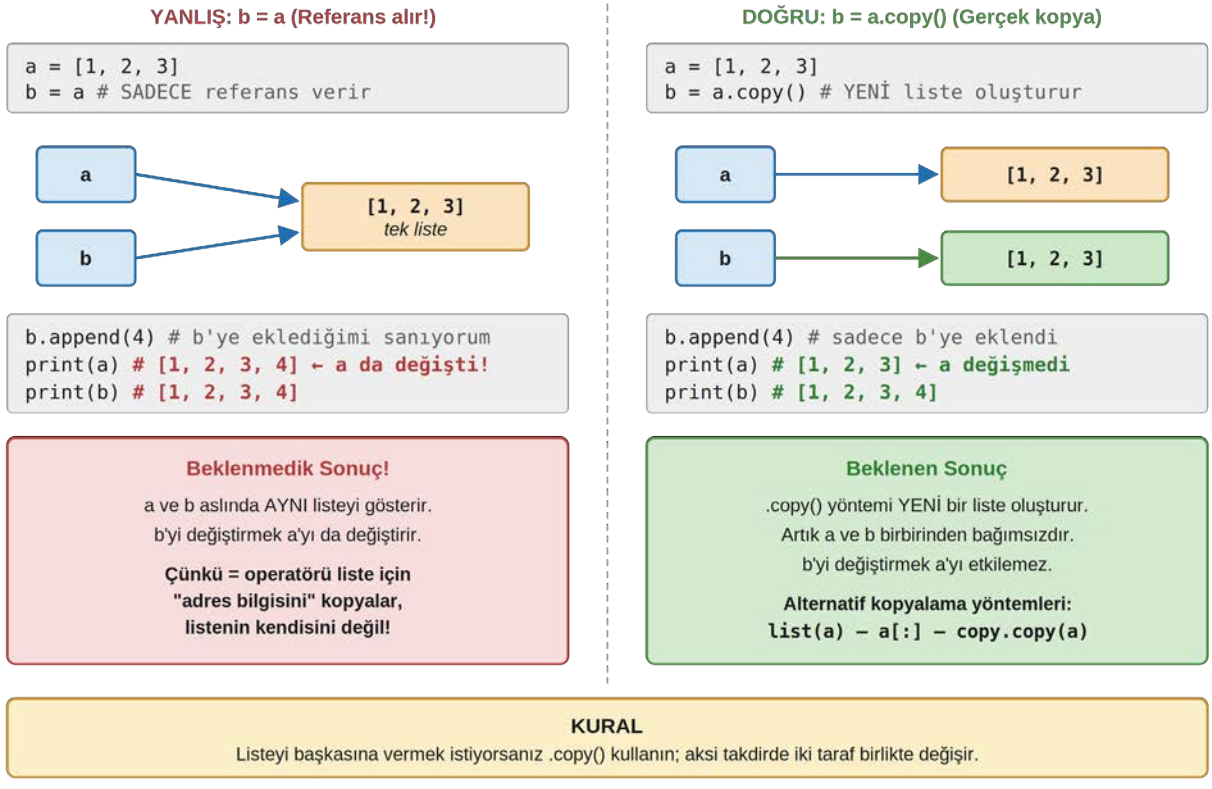
1. Tüm fiyatların toplamını hesaplayınız.

2. 100 TL'den pahalı kaç ürün olduğunu sayınız.

3. sort() kullanmadan, döngü ile karşılaştırarak en yüksek fiyatı bulunuz.

Liste Kopyalama Tuzağı

Görsel 4.3'te iki farklı şekilde liste kopyalama işlemi örneği verilmiştir.



Görsel 4.3: Liste kopyalama işlemleri

19. ÖRNEK UYGULAMA

Amaç: Hatalı ve doğru liste kopyalama yöntemlerini kullanmak

```
1  #YANLIŞ KOPYALAMA
2  notlar=[75,82,68]
3  yedek=notlar
4  yedek.append(100)
5  print("notlar listesi: ",notlar)
6  print("notlar listesinin referans olarak kopyası: ",yedek)
7  #DOĞRU KOPYALAMA
8  notlar=[75,82,68]
9  yedek=notlar.copy()
10 yedek.append(100)
11 print("notlar listesi: ",notlar)
12 print("notlar listesinin copy() fonksiyonu kullanılarak kopyası: ",yedek)
```

EKRAN ÇIKTISI

```
notlar listesi:  [75, 82, 68, 100]
notlar listesinin referans olarak kopyası:  [75, 82, 68, 100]
notlar listesi:  [75, 82, 68]
notlar listesinin copy() fonksiyonu kullanılarak kopyası:  [75, 82, 68, 100]
```

SIRA SİZDE

19. örnek uygulamadaki ekran çıktısının üçüncü satırında iki tane 100 sayısının art arda çıkmasının nedeni nedir?

Çok Boyutlu Listeler

Bir listenin elemanları da liste olabilir. İçinde liste bulunan listelere **çok boyutlu liste** veya **matris** denir. Çok boyutlu listeler, tablo biçimindeki verileri (örneğin bir sınıfın not çizelgesi veya bir satranç tahtası) temsil etmek için kullanılır. Görsel 4.4'te bir sınıfta yer alan öğrencilere ait birden fazla notu içeren bir liste örneği verilmiştir.

Bir sınıfın 4 öğrencisinin 3 sınavdaki notları

	Sınav 1 [0]	Sınav 2 [1]	Sınav 3 [2]
Öğrenci 1 [0]	75	82	90
Öğrenci 2 [1]	68	**74**	85
Öğrenci 3 [2]	92	88	95
Öğrenci 4 [3]	55	60	72

Çift İndeks ile Erişim

notlar[1][1]

→ *Öğrenci 2'nin Sınav 2 notu: 74*

Liste tanımı:

```
notlar = [[75, 82, 90], [68, 74, 85], [92, 88, 95], [55, 60, 72]]
```

Görsel 4.4: Çok boyutlu listenin yapısı ve elemanlarına erişim

Çok boyutlu bir listenin elemanına ulaşmak için **çift indeks** kullanılır: **matris[satır][sütun]**. İlk indeks satırı, ikinci indeks ise o satırdaki sütunu belirtir.

20. ÖRNEK UYGULAMA

Amaç: İç içe döngü ile bir matrisin tüm elemanlarını yazdırmak

Bir matrisin tüm elemanlarına ulaşmak için iç içe iki döngü kullanılır. Dış döngü satırları, iç döngüyse sütunları gezer.

```
1   # 3x3 matris (3 satır, 3 sütun)
2   matris = [
3       [1, 2, 3],
4       [4, 5, 6],
5       [7, 8, 9]
6   ]
7
8   # Eleman erişimi: matris[satir][sutun]
9   print(matris[0][0])     # 1
10  print(matris[1][2])     # 6
```

```
print(matris[2][1])     # 8

# Tüm matrisi yazdırma
for satir in matris:
    for eleman in satir:
        print(eleman, end="")
    print()  # satır sonu
```

EKRAN ÇIKTISI
1
6
8
123
456
789

21. ÖRNEK UYGULAMA

Amaç: Çok boyutlu listede toplam hesaplamak

Çok boyutlu bir listenin tüm elemanlarının toplamı, iç içe döngü ile hesaplanabilir. Aşağıdaki örnek, bir sınıfın not çizelgesindeki tüm notları toplar.

```
not_cizelgesi = [
    [70, 80, 90],
    [60, 65, 70],
    [85, 90, 95]
]

genel_toplam = 0
for ogrenci_notlari in not_cizelgesi:
    for n in ogrenci_notlari:
        genel_toplam = genel_toplam + n

print("Tüm notların toplamı:", genel_toplam)
print("Genel ortalama:", genel_toplam / 9)
```

EKRAN ÇIKTISI
Tüm notların toplamı: 705
Genel ortalama: 78.33333333333333

Dış döngü her öğrencinin not listesini, iç döngü ise o listedeki her notu gezer. Çok boyutlu listeler, tablo biçimindeki verileri işlemek için iç içe döngülerle birlikte kullanılır.

SIRA SİZDE

1. Üç öğrencinin üçer sınav notunu içeren çok boyutlu bir liste oluşturunuz.

Aşağıdaki işlemleri yapınız.

a) İkinci öğrencinin üçüncü notunu yazdırınız.

b) İç içe döngü kullanarak her öğrencinin not ortalamasını hesaplayıp yazdırınız.

2. 2x3 boyutlu (2 satır, 3 sütun) bir matris oluşturunuz. Aşağıdaki işlemleri yapınız.

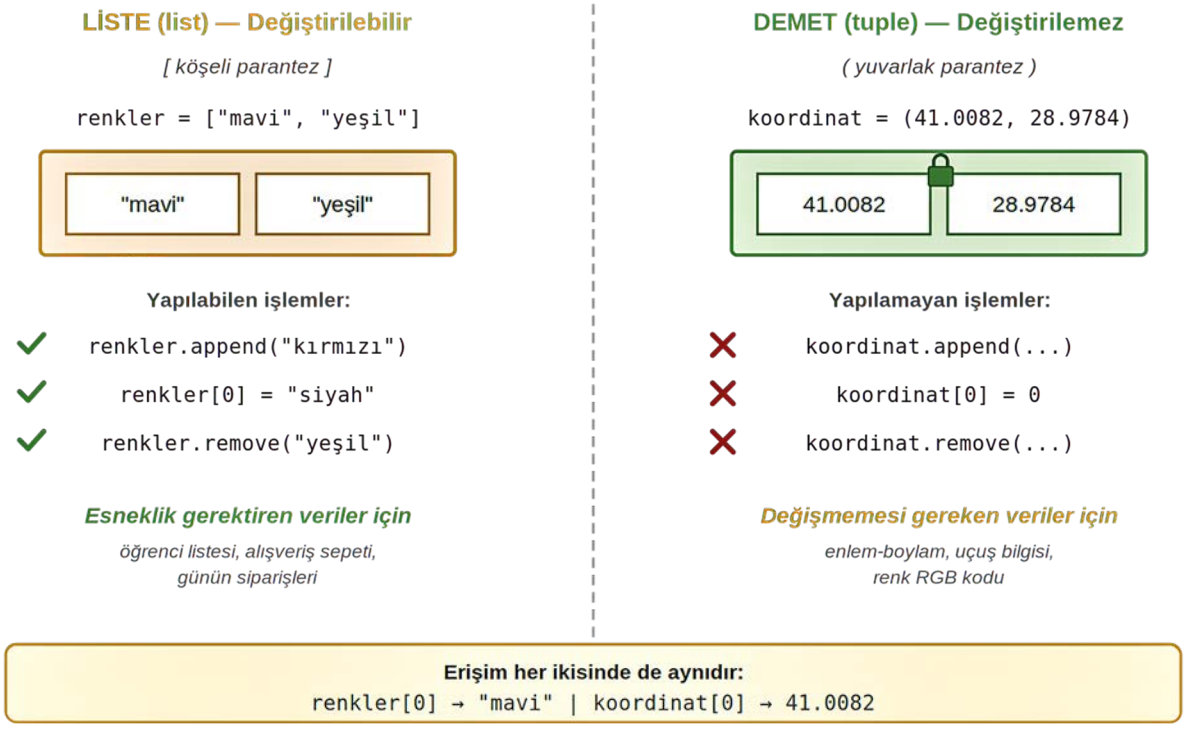
a) Matrisin birinci satırının tüm elemanlarını yazdırınız.

b) İkinci satır, ikinci sütundaki elemanı çift indeks ile yazdırınız.

c) İç içe döngü ile matristeki tüm elemanların toplamını hesaplayınız.

4.2. DEMET VERİ YAPISI

Bazı veriler, programın çalışması boyunca hiç değişmemelidir. Örneğin bir şehrin coğrafi koordinatları, bir uçuşun kalkış-varış bilgileri veya haftanın günleri sabit verilerdir. Bu tür değişmemesi gereken veriler için Python'da demet veri yapısı kullanılır. Liste yapısı ile demet yapısı arasındaki farklar Görsel 4.5'te verilmiştir.



Görsel 4.5: Liste ve demet arasındaki temel farklar

Demet Oluşturma ve Elemanlara Erişim

Demet (tuple), listeye benzeyen ancak **değiştirilemeyen** bir veri yapısıdır. Demet, **normal parantez** içine virgülle ayrılmış elemanlar yazılarak oluşturulur.

```
# Bir demet oluşturma
koordinat = (41.0, 29.0)

# Farklı türde veri içeren demet
ogrenci = ("Ayşe", 15, "9-A")

# Tek elemanlı demet (virgül zorunludur)
tek = (5,)
```

Demet elemanlarına, tıpkı listelerde olduğu gibi indeks ile erişilir. İndeksleme yine 0'dan başlar.

22. ÖRNEK UYGULAMA

Amaç: Bir demet oluşturmak ve elemanlarına erişmek

```
# Tek elemanlı demet (virgül zorunlu!)
tek_eleman = (42,)
# Demet PARANTEZ OLMADAN da oluşturulabilir.
rgb = 255, 128, 0    # Bu da bir demettir!
# Karışık demet
ogrenci = ("Ahmet", 15, 9, 85.5)
# Sayısal demet
koordinat = (41.0082, 28.9784)   # İstanbul'un enlem-boylam değeri
print(koordinat[0])      # 41.0082
print(koordinat[1])      # 28.9784
# Dilimleme de çalışır
print(koordinat[:1])     # (41.0082,)
# DEĞİŞTİRME HATASI
koordinat[0] = 50      # TypeError
```

EKRAN ÇIKTISI

```
41.0082
28.9784
(41.0082,)
  koordinat[0] = 50     # TypeError
  ~~~~~~~~~^^^
TypeError: 'tuple' object does not support item assignment
```

DİKKAT

Tek elemanlı bir demet oluşturulurken elemandan sonra tek = (5,) şeklinde virgül konulması zorunludur. Eğer virgül konmaz ve tek = [5] şeklinde kalırsa Python bunu demet değil normal bir sayı olarak kabul eder. Bu sık yapılan bir hatadır.

23. ÖRNEK UYGULAMA

Amaç: Demet üzerinde dilimleme ve döngü kullanmak

Demetler, listeler gibi dilimleme ve döngü işlemlerini destekler çünkü onlar da sıralı veri yapılarıdır.

```
1  renkler = ("kırmızı", "yeşil", "mavi", "sarı", "mor")
2
3  print("İlk üç renk:", renkler[:3])
4
5  for renk in renkler:
6      print(renk, end=" ")
7  print()
8  print("Demette kaç renk var:", len(renkler))
```

EKRAN ÇIKTISI
İlk üç renk: ('kırmızı', 'yeşil', 'mavi')
kırmızı yeşil mavi sarı mor
Demette kaç renk var: 5

Demet üzerinde dilimleme yapıldığında sonuç yine bir demettir. Demet değiştirilemez olsa da üstünde okuma amaçlı (erişim, dilimleme, döngü) tüm işlemler yapılabilir.

Demetin Değiştirilemezliği

Demet ile liste arasındaki en önemli fark, demetin **değiştirilemez (immutable)** olmasıdır. Bir demet oluşturulduktan sonra elemanları değiştirilemez, yeni eleman eklenemez veya silinemez.

24. ÖRNEK UYGULAMA

Amaç: Demetin değiştirilemez olduğunu gözlemlemek

```
1  renkler = ("kırmızı", "yeşil", "mavi")
2  renkler[0] = "sarı"   # Değiştirme denemesi
```

EKRAN ÇIKTISI
Traceback (most recent call last):
File " C:\kodlar\4\4_24.py ", line 2, in <module>
renkler[0] = "sarı" # Değiştirme denemesi
~~^^^
TypeError: 'tuple' object does not support item assignment

Demetin bir elemanını değiştirmeye çalışmak **TypeError** hatasına yol açar. Bu özellik bir kısıtlama gibi görünse de aslında bir **güvenlik** sağlar ve değişmemesi gereken verilerin yanlışlıkla bozulmasını önler.

Liste ile demet arasındaki temel farklar Tablo 4.2'de verilmiştir.

Tablo 4.2: Liste ve Demet Karşılaştırması

Özellik	Liste (list)	Demet (tuple)
Tanımlama	Köşeli parantez [ ]	Normal parantez ( )
Değiştirilebilir mi?	Evet	Hayır
Eleman eklenebilir mi?	Evet (append, insert)	Hayır
İndeks ile erişim	Var	Var
Hız	Görece yavaş	Görece hızlı
Kullanım amacı	Değişen veriler	Sabit, değişmemesi gereken veriler



BİLGİ KUTUSU

Demetler değiştirilemez olduğu için listelere göre daha hızlı çalışır ve daha az bellek kullanır. Bu nedenle değişmeyecek veriler söz konusu olduğunda demet kullanmak güvenli ve verimli bir tercihtir.



SIRA SİZDE

1. Bir okulun kuruluş bilgilerini içeren bir demet oluşturunuz. okul_bilgisi = ("Cumhuriyet Lisesi", 1995, "Ankara").

Bu demet üzerinde aşağıdaki işlemleri yapınız.

a) Okulun adını ve kuruluş yılını ekrana yazdırınız.

b) Demetin kuruluş yılı elemanını değiştirmeyi deneyiniz ve oluşan hatayı gözlemleyiniz.

c) Bilgilerin bir liste yerine demet içinde tutulmasının neden daha uygun olduğunu açıklayınız.

2. Bir coğrafi konumun enlem ve boylam bilgilerini içeren bir demet oluşturunuz.

konum = (39.92, 32.85).

Bu demet üzerinde aşağıdaki işlemleri yapınız.

a) Demet çözme ile enlem ve boylamı ayrı değişkenlere aktarınız.

b) Her iki değeri de anlamlı bir mesajla ekrana yazdırınız.

c) Bir konumun koordinatlarının neden değiştirilemez (demet) olduğunu açıklayınız.

Demet Kullanım Alanları

Demetler, programlamada birçok durumda kullanılır. Bir demetin elemanları tek bir satırda ayrı değişkenlere aktarılabilir, buna **demet çözme** (unpacking) denir.

25. ÖRNEK UYGULAMA

Amaç: Demet çözme (unpacking) ile elemanları değişkenlere aktarmak

```
1  ogrenci = ("Mehmet", 16, "10-B")
2
3  ad, yas, sinif = ogrenci
4
5  print("Ad:", ad)
6  print("Yaş:", yas)
7  print("Sınıf:" , sinif)
```

EKRAN ÇIKTISI
Ad: Mehmet
Yaş: 16
Sınıf: 10-B

Demet çözme sayesinde bir demetin tüm elemanları, tek bir satırda anlamlı değişken adlarına aktarılır. Bu kullanım kodun okunabilirliğini artırır.

26. ÖRNEK UYGULAMA

Amaç: Demet listesinde döngü ile gezinmek

```
1  sehirler = [
2      ("Ankara", 6),
3      ("İstanbul", 34),
4      ("İzmir", 35)
5  ]
6
7  for ad, plaka in sehirler:
8      print(plaka, "plaka kodu", ad, "şehrine aittir.")
```

EKRAN ÇIKTISI
6 plaka kodu Ankara şehrine aittir.
34 plaka kodu İstanbul şehrine aittir.
35 plaka kodu İzmir şehrine aittir.

Ankara şehrinin plaka bilgisi int veri tipiyle tanımlandığı için 6 olarak görüntülenir. 06 olarak görüntülemek için tüm plaka değerleri " karakteri içinde tanımlanmalıdır.

Döngü her dönüşte bir demeti çözerek elemanlarını ad ve plaka değişkenlerine aktarır. Bu yöntem demet listeleriyle çalışmada en yaygın kullanılan yaklaşımdır.

27. ÖRNEK UYGULAMA

Amaç: Demet ile sabit ayar bilgilerini saklamak

Demetler, bir programın değişmemesi gereken ayar bilgilerini saklamak için idealdir.

```
# Bir oyunun sabit ayarları
EKRAN_BOYUTU = (1920, 1080)
RENK_BEYAZ = (255, 255, 255)

genislik, yukseklik = EKRAN_BOYUTU
print("Ekran genişliği:", genislik)
print("Ekran yüksekliği:", yukseklik)
print("Toplam piksel:", genislik * yukseklik)
```

EKRAN ÇIKTISI
Ekran genişliği: 1920
Ekran yüksekliği: 1080
Toplam piksel: 2073600

Ekran boyutu, renk değerleri gibi sabit bilgiler demet içinde tutulduğunda program çalışırken bunların yanlışlıkla değiştirilmesi engellenir. Bu durum demetin sağladığı güvenliğin pratik bir örneğidir.

SIRA SİZDE

Bir dik üçgenin iki dik kenar uzunluğunu bir demet olarak alan ve hipotenüs uzunluğunu hesaplayan bir program yazınız.

kenarlar = (3, 4) demetini oluşturunuz, demet çözme ile kenarları ayrı değişkenlere aktarınız ve hipotenüsü hesaplayıp yazdırınız (Hipotenüs = İki dik kenarın karelerinin toplamının karekökü dür.).

1. UYGULAMA

Uçuş Bilgi Sistemi Geliştiriyorum

Süre: 40 dakika

Amaç: Her uçuş için sabit olan (uçuş kodu, kalkış yeri, varış yeri ve saat) bilgileri bir demet veri yapısı içinde saklayan uygulamayı geliştirmek

Yönerge: Aşağıdaki adımları takip ederek uygulamayı geliştiriniz.

1. Adım: Editörünüzde **ucus_bilgisi.py** adında yeni bir dosya oluşturunuz. Bir uçuşun uçuş kodu, kalkış yeri, varış yeri, saat gibi hangi bilgilerden oluştuğunu belirleyiniz.

2. Adım: Bir uçuşun bilgilerini bir **demet** içinde saklayınız. Bu bilgilerin uçuş boyunca değişmeyeceği için demet kullanılması uygundur.

```
ucus1 = (“TK2024”, “İstanbul”, “Ankara”, “14:30”)
```

3. Adım: Demet çözme (unpacking) kullanarak uçuş bilgilerini ayrı değişkenlere aktarınız ve düzenli bir biçimde ekrana yazdırınız.

```
kod, kalkis, varis, saat = ucus1
print(“Uçuş Kodu:”, kod)
print(“Kalkış:”, kalkis)
print(“Varış:”, varis)
print(“Saat:”, saat)
```

4. Adım: En az üç farklı uçuş için demet oluşturunuz ve bu demetleri bir liste içinde toplayınız. Değişmeyen uçuş bilgileri demetle ve uçuşların listesi liste veri yapısıyla yönetilmiş olur.

```
ucuslar = [
    (“TK2024”, “İstanbul”, “Ankara”, “14:30”),
    (“TK3015”, “İzmir”, “Antalya”, “09:15”),
    (“TK4102”, “Ankara”, “Trabzon”, “18:45”)
```

5. Adım: Bir **for döngüsü** kullanarak listedeki tüm uçuşları sırayla ekrana yazdırınız.

6. Adım: Bir uçuşun bilgisini (örneğin varış yerini) değiştirmeyi deneyiniz ve demetin değiştirilemez olması nedeniyle oluşan hatayı gözlemleyiniz. Bu durumun, uçuş bilgilerinin yanlışlıkla bozulmasını nasıl önlediğini açıklayınız.

7. Adım: Tamamlanan programınızı README dosyasıyla birlikte dijital deponuza yükleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

SIRA SİZDE

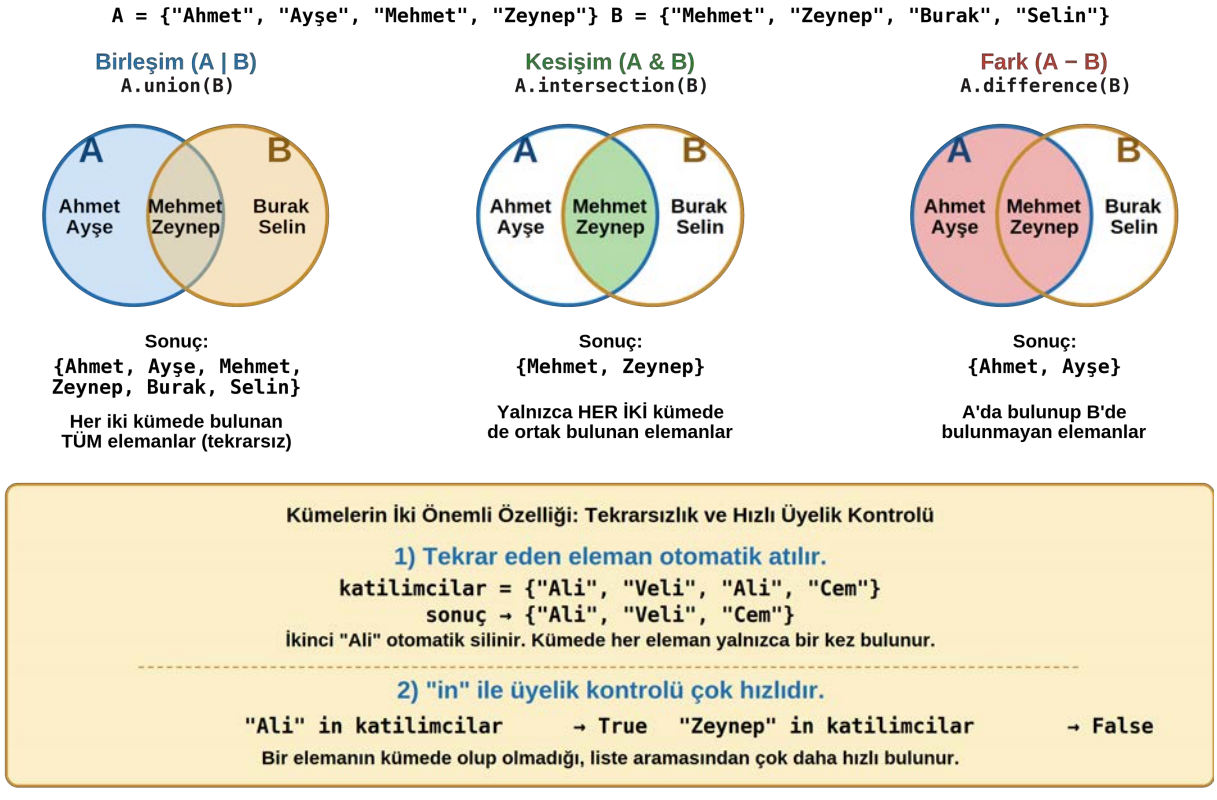
1. Uygulamada geliştirdiğiniz uçuş programına yeni bir özellik ekleyiniz. Kullanıcıdan bir uçuş kodu alıp uçuş listesinde bu koda sahip uçuşu bulunuz. Bulduğunuz uçuş bilgilerini ekrana yazdıran bir program yazınız. (Uçuş listesini for döngüsü ile gezerek her uçuşun ilk elemanını (uçuş kodu) kullanıcının girdiği uçuş kodu değeriyle karşılaştırınız.)

2. Aşağıdaki durumları inceleyiniz. Her veri için liste mi yoksa demet mi kullanılması gerektiğini gerekçesiyle açıklayınız.

a) Bir öğrencinin not listesi (Dönem boyunca yeni notlar eklenecek.)
b) Bir yılın 12 ayının adları (Hiç değişmez.)
c) Bir alışveriş sepeti (Ürün eklenip çıkarılacaktır.)
ç) Bir dik üçgenin sabit kenar oranları

4.3. KÜME VERİ YAPILARI

Bazı durumlarda bir koleksiyondaki elemanların yalnızca bir kez bulunması gerekir. Örneğin bir çekilişe katılanlar listesinde aynı kişi iki kez yer almamalı veya ziyaret edilen şehirler listesinde tekrar eden şehirler bulunmamalıdır. Bu tür benzersiz (tekrarsız) veriler için Python'da küme veri yapısı kullanılır. Küme yapısının tanımlanması ve farklı kümelerin birbirleriyle olan işlemleri Görsel 4.6'da gösterilmiştir.



Görsel 4.6: Küme tanımlama ve kümeler arası temel işlemler

Küme (set), tekrarlanmayan ve **sırasız** elemanlardan oluşan bir veri yapısıdır. Küme, **süslü parantez** içine virgülle ayrılan elemanlar yazılarak oluşturulur.

28. ÖRNEK UYGULAMA

Amaç: Küme tanımlayarak yazdırmak

```
# Tekrar eden elemanlar otomatik silinir
urunler = {"ekmek", "süt", "ekmek", "yumurta", "süt"}
print(urunler)

# Boş küme
bos_kume = set()

# Listeyi kümeye dönüştürme - tekrarları kaldırmak için
notlar = [75, 82, 75, 90, 68, 82]
benzersiz = set(notlar)
print(benzersiz)    # {75, 82, 90, 68}
```

EKRAN ÇIKTISI

{'süt', 'yumurta', 'ekmek'}

{82, 75, 68, 90}

Kümenin iki temel özelliği vardır: Elemanları **benzersizdir**, aynı eleman iki kez bulunamaz, küme **sırasız bir yapıdadır.** Bu nedenle elemanların belirli bir indeksi yoktur.

Küme oluşturulurken tekrarlı elemanlar **otomatik olarak** elenir. Bu özellik, bir listedeki tekrarları temizlemenin en pratik yoludur.

DİKKAT

Küme sırasız olduğu için elemanlarına indeks ile erişilemez. kume[0] gibi bir ifade TypeError hatasına yol açar. Boş bir küme oluşturmak için {} değil set() kullanılmalıdır çünkü {} boş bir sözlük oluşturur.

29. ÖRNEK UYGULAMA

Amaç: Kümede döngü ile gezinmek ve üyelik kontrolü yapmak

Küme sırasız olsa da elemanları üzerinde döngüyle gezilebilir ve in operatörü kullanılarak üyelik kontrolü yapılabilir.

```
diller = {"Python", "Java", "C++", "JavaScript"}

for dil in diller:
    print(dil)

print("---")
print("Python kümede var mı?", "Python" in diller)
print("Ruby kümede var mı?", "Ruby" in diller)
```

EKRAN ÇIKTISI

```
JavaScript
Java
C++
Python
---
Python kümede var mı? True
Ruby kümede var mı? False
```

Küme sırasız olduğu için elemanların yazdırılma sırası, ekleme sırasından farklı olabilir. in operatörü bir elemanın kümede olup olmadığını hızlıca kontrol eder.

30. ÖRNEK UYGULAMA

Amaç: Bir metindeki farklı harfleri küme ile bulmak

Küme, bir metindeki farklı (benzersiz) karakterleri bulmak için kullanılabilir.

```
1  kelime = "programlama"
2
3  farkli_harfler = set(kelime)
4  print("Kelimedeki farklı harfler:", farkli_harfler)
5  print("Farklı harf sayısı:", len(farkli_harfler))
```

EKRAN ÇIKTISI

```
Kelimedeki farklı harfler: {'o', 'l', 'r', 'a', 'g', 'p', 'm'}
Farklı harf sayısı: 7
```

"programlama" kelimesi 11 harften oluşur ama yalnızca **7 farklı** harf içerir. Küme, tekrar eden harfleri (a, r, m) otomatik eleyerek benzersiz harfleri verir.

Küme Metot ve İşlemleri

Kümeler üzerinde eleman ekleme, silme ve çeşitli işlemler için metotlar bulunur. En sık kullanılan küme metotları Tablo 4.3'te verilmiştir.

Tablo 4.3: Sık Kullanılan Küme Metotları

Metot	Görevi	Örnek Kullanım
add(x)	Kümeye x elemanını ekler.	kume.add(5)
remove(x)	Kümeden x elemanını siler.	kume.remove(5)
discard(x)	x varsa siler yoksa hata vermez.	kume.discard(5)
union(k)	İki kümenin birleşimini döndürür.	k1.union(k2)
intersection(k)	İki kümenin kesişimini döndürür.	k1.intersection(k2)
difference(k)	İki kümenin farkını döndürür.	k1.difference(k2)

31. ÖRNEK UYGULAMA

Amaç: Kümeye eleman eklemek ve silmek

```
1  takim = {"Ali", "Veli", "Ayşe"}
2  print("Başlangıç:", takim)
3
4  takim.add("Mehmet")
5  print("Ekleme sonrası:", takim)
6
7  takim.discard("Veli")
8  print("Silme sonrası:", takim)
9
10 takim.add("Ali")   # Zaten var, tekrar eklenmez
11 print("Tekrar ekleme sonrası:", takim)
```

EKRAN ÇIKTISI
Başlangıç: {'Ali', 'Veli', 'Ayşe'}
Ekleme sonrası: {'Ali', 'Veli', 'Ayşe', 'Mehmet'}
Silme sonrası: {'Ali', 'Ayşe', 'Mehmet'}
Tekrar ekleme sonrası: {'Ali', 'Ayşe', 'Mehmet'}

"Ali" zaten kümede bulunduğu için tekrar eklendiğinde küme değişmedi. Küme, elemanların benzersizliğini otomatik olarak korur.

Matematikteki küme işlemleri, programlama dillerinde de söz diziminde de benzer şekilde uygulanabilir. Bu işlemlerde kullanılacak operatörler Tablo 4.4'te kullanım amaçlarıyla birlikte verilmiştir.

Tablo 4.4: Kümeler Arası İşlemler

İşlem	Operatör	Görevi	Örnek Kullanım
Birleşim	|	union()	Her iki kümenin tüm elemanları
Kesişim	&	intersection()	Ortak elemanlar
Fark	-	difference()	Birinde olup diğerinde olmayanlar
Simetrik Fark	^	symmetric_difference()	Yalnız birinde olanlar

32. ÖRNEK UYGULAMA

Amaç: Kümeler arasında işlemler uygulamak

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

print(A | B)    # {1, 2, 3, 4, 5, 6}   birleşim
print(A & B)    # {3, 4}                kesişim
print(A - B)    # {1, 2}                A'da olup B'de olmayan
print(A ^ B)    # {1, 2, 5, 6}         simetrik fark
```

EKRAN ÇIKTISI
{1, 2, 3, 4, 5, 6}
{3, 4}
{1, 2}
{1, 2, 5, 6}

33. ÖRNEK UYGULAMA

Amaç: : Küme işlemlerinden birleşim, kesişim ve farkı ayırt etmek

Kümeler, matematikteki küme işlemlerini (birleşim, kesişim, fark) desteklediği için çok kullanışlıdır.

```
matematik = {"Ali", "Ayşe", "Veli"}
fizik = {"Ayşe", "Veli", "Zeynep"}

print("Her iki kursa da gidenler:", matematik.intersection(fizik))
print("Kurslardan birine gidenler:", matematik.union(fizik))
print("Yalnızca matematiğe gidenler:", matematik.difference(fizik))
```

EKRAN ÇIKTISI
Her iki kursa da gidenler: {'Ayşe', 'Veli'}
Kurslardan birine gidenler: {‘Ayşe’, ‘Veli’, ‘Zeynep’, ‘Ali’}
Yalnızca matematiğe gidenler: {‘Ali’}

34. ÖRNEK UYGULAMA

Amaç: Küme işlemleriyle ortak ve farklı elemanları bulmak

Küme işlemleri, gerçek hayat problemlerinde sıkça kullanılır. Aşağıdaki örnek iki gün süren bir etkinliğe katılanları analiz eder.

```
1  birinci_gun = {"Ali", "Ayşe", "Can", "Ece"}
2  ikinci_gun = {"Ayşe", "Can", "Deniz", "Fatma"}
3
4  her_iki_gun = birinci_gun.intersection(ikinci_gun)
5  print("Her iki gün de katılanlar:", her_iki_gun)
6
7  sadece_birinci = birinci_gun.difference(ikinci_gun)
8  print("Sadece birinci gün katılanlar:", sadece_birinci)
9
10 toplam_katilimci = birinci_gun.union(ikinci_gun)
11 print("Toplam farklı katılımcı sayısı:", len(toplam_katilimci))
```

EKRAN ÇIKTISI

```
Her iki gün de katılanlar: {'Can', 'Ayşe'}
Sadece birinci gün katılanlar: {'Ece', 'Ali'}
Toplam farklı katılımcı sayısı: 6
```

Küme işlemleri sayesinde iki grup arasındaki ortak elemanlar, farklılıklar ve toplam benzersiz eleman sayısı tek bir satırda bulunabilir. Bu işlemlerin listelerle yapılması zordur.

BİLGİ KUTUSU

Bir elemanın küme içinde olup olmadığı **in operatörü** ile çok hızlı biçimde kontrol edilir. "Ali" in takim ifadesi True veya False döndürür. Kümeler bu kontrolü listelere göre çok daha hızlı yapar. Büyük veri kümelerinde üyelik kontrolü için kümeler tercih edilir.

35. ÖRNEK UYGULAMA

Amaç: : Küme ile bir listedeki tekrarları saymak

Küme, bir listede kaç farklı eleman olduğunu bulmak için pratik bir yol sunar.

```
1  oylar = ["evet", "hayır", "evet", "evet", "hayır", "çekimser"]
2
3  farkli_secenekler = set(oylar)
4  print("Kullanılan farklı oylar:", farkli_secenekler)
5  print("Kaç farklı seçenek var:", len(farkli_secenekler))
```

EKRAN ÇIKTISI

```
Kullanılan farklı oylar: {'evet', 'hayır', 'çekimser'}

Kaç farklı seçenek var: 3
```

Listede altı oy olmasına rağmen yalnızca üç farklı seçenek kullanılmıştır. Küme, tekrarları otomatik olarak elediği için farklı eleman sayısını bulmanın en kolay yoludur.

SIRA SİZDE

1. İki farklı günde kütüphaneye gelen öğrencilerin adlarını iki küme olarak oluşturunuz.

pazartesi = {“Ali”, “Ayşe”, “Can”, “Deniz”}

sali = {“Ayşe”, “Deniz”, “Ece”, “Ali”}

Küme işlemlerini kullanarak aşağıdaki soruları cevaplayınız.

a) Her iki gün de gelen öğrenciler kimlerdir?

b) En az bir gün gelen tüm öğrenciler kimlerdir?

c) Yalnızca pazartesi gelenler kimlerdir?

2. Bir spor okulunda futbol ve basketbol kurslarına kayıtlı öğrenciler iki küme olarak verilmiştir. Aşağıdaki soruları küme işlemleriyle cevaplayan bir program yazınız.

a) Futbol ve basketbol oynayan öğrenciler kimlerdir?

b) Yalnızca futbol oynayan öğrenciler kimlerdir?

c) İki kursa kayıtlı toplam kaç farklı öğrenci vardır?

ç) Kendi belirlediğiniz örnek öğrenci adlarıyla programı test ediniz.

2. UYGULAMA

Benzersiz Ürün Kataloğu Geliştiriyorum

Süre: 40 dakika

Amaç: Bir markette bulunan ve satılan ürünleri (tekrarlı elemanları içermez) takip ederek stok durumunu kontrol eden uygulamayı geliştirmek

Yönerge: Aşağıdaki adımları takip ederek uygulamayı geliştiriniz.

1. Adım: Editörünüzde **market_stok.py** adında yeni bir dosya oluşturunuz. Problemi inceleyiniz. Bir markette aynı ürün stok listesinde yalnızca bir kez yer almalıdır.

2. Adım: Markette bulunan ürünleri bir **küme** olarak tanımlayınız.

```
stok = {“ekmek”, “süt”, “yumurta”, “peynir”, “çay”}
```

3. Adım: Markete yeni gelen ürünleri add() metodu ile stoğa ekleyiniz. Aynı üründen tekrar gelirse kümenin onu otomatik olarak elediğini gözlemleyiniz.

```
stok.add("şeker")
stok.add("ekmek")   # Zaten var, tekrar eklenmez
print("Güncel stok:", stok)
```

4. Adım: Satılan (stoğu biten) bir ürünü discard() metoduyla stoktan çıkarınız.

5. Adım: Kullanıcıdan bir ürün adı alınız ve in operatörüyle bu ürünün stokta olup olmadığını kontrol ediniz.

```
urun = input("Aranan ürün: ")
if urun in stok:
    print(urun, "stokta mevcut.")
else:
    print(urun, "stokta yok.")
```

6. Adım: İki farklı şubenin ürün kümelerini oluşturarak küme işlemlerini uygulayınız. Her iki şubede de bulunan ürünleri (kesişim) ve toplam ürün çeşidini (birleşim) bulunuz.

7. Adım: Tamamlanan programınızı README dosyasıyla birlikte dijital deponuza yükleyiniz

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

SIRA SİZDE

1. Bir sınıftaki öğrencilerin seçtiği kulüpleri tutan bir program yazınız. Üç öğrencinin seçtiği kulüpleri üç ayrı küme olarak oluşturunuz.

Aşağıdaki işlemleri yapınız.

a) Üç öğrencinin ortak seçtiği kulüpleri bulunuz.

b) Seçilen tüm kulüplerin listesini (tekrarsız) oluşturunuz.

c) Toplam kaç farklı kulüp seçildiğini len() fonksiyonuyla bulunuz.

2. Bir okul gezisine katılmak isteyen öğrencilerin adları, farklı sınıflardan toplandığı için tekrarlı olarak bir listede toplanmıştır.

basvuranlar = [“Ali”, “Ece”, “Ali”, “Can”, “Ece”, “Deniz”, “Ali”]

Aşağıdaki işlemleri yapınız.

a) Bu listeyi kümeye çevirerek tekrarsız öğrenci adlarını elde ediniz.

b) Kaç farklı öğrencinin başvurduğunu bulunuz.

c) “Ali” adlı öğrencinin başvurup başvurmadığını in operatörü ile kontrol ediniz.

4.4. SÖZLÜK VERİ YAPISI

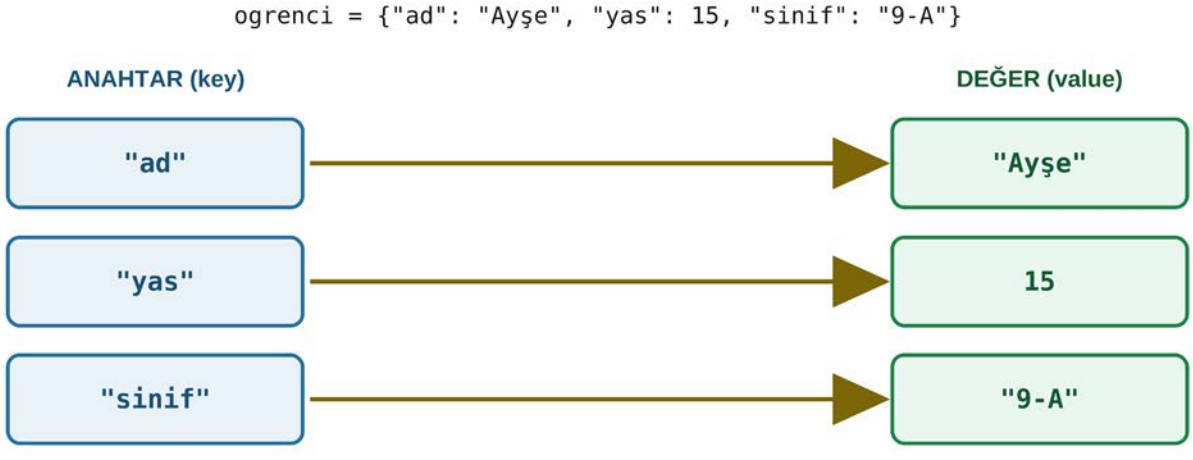
Liste, demet ve küme veri yapılarında elemanlara ya indeks ile ya da hiç sıra olmadan erişilebilir ancak bazı verileri bir etiketle (isimle) saklamak ve o etiket aracılığıyla verilere erişmek gerekebilir. Örneğin bir öğrencinin “adı”, “yaşı”, “sınıfı” gibi bilgilerini etiketleriyle birlikte tutmak istenebilir. **Sözlük (dictionary)**, verileri **anahtar-değer** çiftleri biçiminde saklayan bir veri yapısıdır.

Sözlük Oluşturma ve Elemanlara Erişim

Her veri (değer), kendisine ait bir etiketle (anahtar) birlikte tutulur. Sözlük, **süslü parantezler** içine **anahtar: değer** çiftleri yazılarak oluşturulur.

```
1  # Bir sözlük oluşturma
2  ogrenci = {"ad": "Ayşe", "yas": 15, "sinif": "9-A"}
3
4  # Boş bir sözlük
5  bos_sozluk = {}
```

Sözlüğün çalışma mantığı, gerçek bir sözlük kitabının çalışma mantığına benzer, bir kelimeye (anahtar) bakarak onun anlamını (değer) bulursunuz. Anahtar-değer ilişkisi Görsel 4.7’de gösterilmiştir.



Görsel 4.7: Sözlük, anahtar-değer ilişkisi

Sözlükte bir değere ulaşmak için o değerin anahtarı kullanılır. Listelerdeki sayısal indeksler yerine anahtarlar kullanılır.

36. ÖRNEK UYGULAMA

Amaç: Bir sözlük oluşturmak ve değerlerine anahtarla erişmek

```
1   # Boş sözlük
2   sozluk = {}
3   ogrenci = {
4       "ad": "Ayşe", "yas": 15, "sinif": "9-A","ortalama": 85.5
5   }
6   print(ogrenci["ad"])
7   print(ogrenci["yas"])
8   # Değer değiştirme
9   ogrenci["yas"] = 16
10  # Yeni anahtar-değer ekleme
11  ogrenci["okul"] = "Atatürk Lisesi"
12  print(ogrenci)
13  # Anahtarı silme
14  del ogrenci["sinif"]
15  print(ogrenci)
```

EKRAN ÇIKTISI
Ayşe
15
{'ad': 'Ayşe', 'yas': 16, 'sinif': '9-A', 'ortalama': 85.5, 'okul': 'Atatürk Lisesi'}
{'ad': 'Ayşe', 'yas': 16, 'ortalama': 85.5, 'okul': 'Atatürk Lisesi'}

37. ÖRNEK UYGULAMA

Amaç: Liste ve sözlük kullanarak aynı veriyi saklama yöntemlerini karşılaştırmak

```
# Liste ile – hangi indeks neydi?
ogrenci_liste = ["Ayşe", 15, "9–A"]
print(ogrenci_liste[1])      # 1 neyi ifade ediyor?

# Sözlük ile – anahtarlar anlamı açıkça gösterir
ogrenci_sozluk = {"ad": "Ayşe", "yas": 15, "sinif": "9–A"}
print(ogrenci_sozluk["yas"])  # yaş olduğu açıkça belli
```

EKRAN ÇIKTISI
15
15

İki yöntemle de aynı çıktı üretilir ancak listedeki **ogrenci_liste[1]** ifadesiyle yaş gösterilir. Sözlükte ise **ogrenci_sozluk["yas"]** ifadesiyle hangi veriye erişildiği gösterilir. Bu nedenle etiketli veriler için sözlük veri yapısı kullanılır.

Sözlükler **değiştirilebilir** yapılardır. Var olan bir anahtarın değeri güncellenebilir veya yeni bir anahtar değer çifti eklenebilir.

38. ÖRNEK UYGULAMA

Amaç: Sözlüğe yeni eleman eklemek ve var olan değeri güncellemek

```
1  ogrenci = {"ad": "Ayşe", "yas": 15}
2  print("Başlangıç:", ogrenci)
3
4  ogrenci["yas"] = 16            # var olan değeri güncelle
5  print("Güncelleme sonrası:", ogrenci)
6
7  ogrenci["sinif"] = "9-A"       # yeni anahtar-değer ekle
8  print("Ekleme sonrası:", ogrenci)
```

EKRAN ÇIKTISI
Başlangıç: {'ad': 'Ayşe', 'yas': 15}
Güncelleme sonrası: {'ad': 'Ayşe', 'yas': 16}
Ekleme sonrası: {'ad': 'Ayşe', 'yas': 16, 'sinif': '9-A'}

DİKKAT

Bir sözlükte var olmayan bir anahtara erişmeye çalışmak KeyError hatasına yol açar. Bir anahtarın sözlükte olup olmadığı in operatörü ile kontrol edilebilir veya değere güvenli biçimde erişmek için get() metodu kullanılabilir.

BİLGİ KUTUSU

Sözlük yapısında anahtar (key) olarak kullanılacak verilerin **değiştirilemez** (immutable) veri tiplerinden olması zorunludur.

Programlama sürecinde sıklıkla kullandığımız metinler (str), sayılar (int, float) ve demetler (tuple) değiştirilemez yapıda oldukları için sözlüklerde anahtar olarak güvenle kullanılır ancak listeler (list) **değiştirilebilir** (mutable) yapıda olduklarından bir sözlüğe asla anahtar olarak atanamazlar. Listeyi anahtar olarak kullanmaya çalışmak programın hata vermesine yol açar.

Sözlükteki değerler (value) kısmında ise herhangi bir veri tipi kısıtlaması yoktur. Değer olarak bir sayı, metin, liste, demet hatta başka bir sözlük yapısı kullanılabilir.

39. ÖRNEK UYGULAMA

Amaç: Sözlükte anahtar varlığını kontrol etmek

Bir anahtarın sözlükte olup olmadığı in operatörü ile kontrol edilir. Bu yöntem, KeyError hatasını önlemenin güvenli bir yoludur.

```
1  telefon_rehberi = {"Ali": "0532...", "Ayşe": "0541...", "Veli": "0505..."}
2
3  aranan = "Ayşe"
4  if aranan in telefon_rehberi:
5      print(aranan, "->", telefon_rehberi[aranan])
6  else:
7      print(aranan, "rehberde kayıtlı değil.")
8
9  aranan2 = "Mehmet"
10 if aranan2 in telefon_rehberi:
11     print(aranan2, "->", telefon_rehberi[aranan2])
12 else:
13     print(aranan2, "rehberde kayıtlı değil.")
```

EKRAN ÇIKTISI

```
Ayşe -> 0541...

Mehmet rehberde kayıtlı değil.
```

in operatörü, sözlükte yalnızca **anahtarları** kontrol eder. Bu kontrol sayesinde var olmayan bir anahtara erişmeden önce onun varlığından emin olunur ve programın çökmesi önlenir.

SIRA SİZDE

Sevdiğiniz bir kitabın bilgilerini içeren bir sözlük oluşturunuz. Sözlükte "ad", "yazar", "sayfa_sayisi", "yayin_yili" anahtarları bulunsun. Buna göre aşağıdaki işlemleri yapınız.

1. Kitabın adını ve yazarını ekrana yazdırınız.

2. Sözlüğe "tur" adında yeni bir anahtar-değer ekleyiniz.

3. Sayfa sayısı bilgisini güncelleyiniz.

4. "fiyat" anahtarının sözlükte olup olmadığını in operatörüyle kontrol ediniz.

Sözlük Metotları

Sözlükler üzerinde işlem yapmak için çeşitli metotlar bulunur. En çok kullanılan sözlük metotları Tablo 4.5'te verilmiştir.

Tablo 4.5: Sık Kullanılan Sözlük Metotları

Metot	Görevi	Örnek Kullanım
keys()	Tüm anahtarları döndürür.	sozluk.keys()
values()	Tüm değerleri döndürür.	sozluk.values()
items()	Anahtar-değer çiftlerini döndürür.	sozluk.items()
get(k,v)	k anahtarının değerini güvenli döndürür. Eğer k değeri yoksa v değerini döndürür.	sozluk.get("ad","yok")
pop(k)	k anahtarını siler ve değerini döndürür.	sozluk.pop("yas")
update(d)	Sözlüğe başka bir sözlüğü ekler.	sozluk.update(d2)

40. ÖRNEK UYGULAMA

Amaç: keys(), values() ve items() metotlarını kullanmak

```
1  urun = {"ad": "Defter", "fiyat": 30, "stok": 100}
2
3  print("Anahtarlar:", urun.keys())
4  print("Değerler:", urun.values())
5  print("Çiftler:", urun.items())
```

EKRAN ÇIKTISI

```
Anahtarlar: dict_keys(['ad', 'fiyat', 'stok'])
Değerler: dict_values(['Defter', 30, 100])
Çiftler: dict_items([('ad', 'Defter'), ('fiyat', 30), ('stok', 100)])
```

41. ÖRNEK UYGULAMA

Amaç: get() metodu ile güvenli erişim sağlamak

get() metodu, var olmayan bir anahtar için hata vermek yerine None veya belirtilen varsayılan değeri döndürür.

```
1  ogrenci = {"ad": "Mehmet", "yas": 16}
2
3  print("Var olan anahtar:", ogrenci.get("ad"))
4  print("Olmayan anahtar:", ogrenci.get("telefon"))
5  print("Varsayılan değerle:", ogrenci.get("telefon", "Kayıt yok"))
```

EKRAN ÇIKTISI
Var olan anahtar: Mehmet
Olmayan anahtar: None
Varsayılan değerle: Kayıt yok



BİLGİ KUTUSU

get() metodu, bir anahtarın var olup olmadığından emin olunmadığı durumlarda kullanılır. Köşeli parantezle erişim (sozluk["anahtar"]), anahtar yoksa programı çökertir. get() metoduyla programın çökmesi önlenir ve güvenli bir erişim sağlanır.



42. ÖRNEK UYGULAMA

Amaç: Sözlükten eleman silmek ve güncellemek

pop() metoduyla bir anahtar-değer çifti sözlükten silinir, update() metoduyla bir sözlüğe başka bir sözlüğün elemanları eklenir.

```
1   urun = {"ad": "Defter", "fiyat": 30, "stok": 100}
2
3   # pop() ile eleman silme
4   silinen = urun.pop("stok")
5   print("Silinen değer:", silinen)
6   print("Güncel sözlük:", urun)
7
8   # update() ile sözlüğe ekleme
9   urun.update({"renk": "mavi", "fiyat": 35})
10  print("Güncelleme sonrası:", urun)
```

EKRAN ÇIKTISI
Silinen değer: 100
Güncel sözlük: {'ad': 'Defter', 'fiyat': 30}
Güncelleme sonrası: {'ad': 'Defter', 'fiyat': 35, 'renk': 'mavi'}

update() metoduyla var olan anahtarın (fiyat) değeri güncellenir, var olmayan anahtar sözlüğe eklenir.

SIRA SİZDE

1. Ülkelerin başkentlerini tutan bir sözlük oluşturunuz. baskentler = {"Türkiye": "Ankara", "Fransa": "Paris", "Japonya": "Tokyo"}.

Aşağıdaki işlemleri yapınız.

a) keys() metodunu kullanarak tüm ülkeleri yazdırınız.

b) values() metodunu kullanarak tüm başkentleri yazdırınız.

c) get() metodunu kullanarak "Almanya" anahtarını "Bilinmiyor" varsayılan değeriyle sorgulayınız.

2. Bir öğrencinin derslerini ve aldığı notları içeren bir sözlük oluşturunuz. dersler = {"Matematik": 85, "Fizik": 70, "Kimya": 90}.

Aşağıdaki işlemleri yapınız.

a) update() metodu ile sözlüğe "Biyoloji" dersini ve notunu ekleyiniz.

b) pop() metodu ile "Fizik" dersini sözlükten siliniz.

c) items() metodu ve döngü ile kalan tüm ders ve notları yazdırınız.

ç) Her işlemden sonra sözlüğün son hâlini ekrana yazdırınız.

Sözlükte Döngüyle Gezinme

Bir sözlüğün tüm anahtar-değer çiftleri üzerinde işlem yapmak için for döngüsü kullanılır. items() metodu ile her dönüşte anahtar ve değere aynı anda erişilebilir.

43. ÖRNEK UYGULAMA

Amaç: Bir sözlüğün tüm elemanlarını döngü ile yazdırmak

```
fiyatlar = {"ekmek": 8, "süt": 25, "yumurta": 60}

# Sadece anahtarlar üzerinde
for urun in fiyatlar:
    print(urun)

# Sadece değerler üzerinde
for fiyat in fiyatlar.values():
    print(fiyat)

# Hem anahtar hem değer
for urun, fiyat in fiyatlar.items():
    print(f"{urun}: {fiyat} TL")
```

EKRAN ÇIKTISI
ekmek
süt
yumurta
8
25
60
ekmek: 8 TL
süt: 25 TL
yumurta: 60 TL

keys() fonksiyonuyla yalnızca anahtarlar, **values()** fonksiyonuyla yalnızca değerler ve **items()** fonksiyonuyla her ikisi birlikte verilir. Hangisinin kullanılacağı, döngüde neye ihtiyaç duyulduğuna bağlıdır.

44. ÖRNEK UYGULAMA

Amaç: Sözlükte döngü ile hesaplama yapmak

Bir sözlükteki değerler üzerinde döngü ile hesaplama yapılabilir.

```
1  sepet = {"defter": 30, "kalem": 15, "silgi": 5, "cetvel": 20}
2  toplam = 0
3
4  for fiyat in sepet.values():
5      toplam = toplam + fiyat
6
7  print("Sepetteki ürün sayısı:", len(sepet))
8  print("Toplam tutar:", toplam, "TL")
```

EKRAN ÇIKTISI
Sepetteki ürün sayısı: 4
Toplam tutar: 70 TL

45. ÖRNEK UYGULAMA

Amaç: Sözlükte koşullu işlem yapmak ve en büyük değeri bulmak

Bir sözlükteki değerler üzerinde koşul kullanarak belirli kayıtlar bulunabilir. Aşağıdaki örnek, stoğu en yüksek ürünü bulur.

```
1   stok = {"ekmek": 50, "süt": 120, "yumurta": 90, "peynir": 30}
2
3   en_cok_urun = ""
4   en_cok_adet = 0
5
6   for urun, adet in stok.items():
7       if adet > en_cok_adet:
8           en_cok_adet = adet
9           en_cok_urun = urun
10
11  print("En çok stoğu olan ürün:", en_cok_urun)
12  print("Stok adedi:", en_cok_adet)
```

EKRAN ÇIKTISI

```
En çok stoğu olan ürün: süt
Stok adedi: 120
```

46. ÖRNEK UYGULAMA

Amaç: Sözlükle sayım (frekans) yapmak

Sözlükler, bir veride her elemanın kaç kez geçtiğini saymak için çok kullanışlıdır. Aşağıdaki örnek, bir listede her rengin kaç kez tekrarlandığını sayar.

```
1   renkler = ["mavi", "kırmızı", "mavi", "yeşil", "mavi", "kırmızı"]
2   sayim = {}
3
4   for renk in renkler:
5       if renk in sayim:
6           sayim[renk] = sayim[renk] + 1
7       else:
8           sayim[renk] = 1
9
10  print("Renk sayımı:", sayim)
```

EKRAN ÇIKTISI

```
Renk sayımı: {'mavi': 3, 'kırmızı': 2, 'yeşil': 1}
```

Program, her rengi bir anahtar olarak kullanır ve o rengin kaç kez geçtiğini değer olarak tutar. Renk daha önce görülmüşse sayısı 1 artırılır, daha önce görülmemişse 1 ile başlatılır. Bu yöntem sözlüğün en güçlü kullanım alanlarından biridir.

Döngü, sözlüğün her anahtar-değer çiftini gezerek en yüksek değeri ve ona ait anahtarı bulur. Bu yöntem, sözlükte arama ve karşılaştırma yapmanın temel kalıbıdır.

SIRA SİZDE

Bir sınıftaki öğrencilerin notlarını içeren bir sözlük oluşturunuz. notlar = {"Ali": 70, "Ayşe": 90, "Veli": 55, "Ece": 85}.

Aşağıdaki işlemleri yapınız.

1. Tüm öğrencilerin ad ve notlarını alt alta yazdırınız.

2. Sınıfın not ortalamasını hesaplayınız.

3. Notu 80'in üzerinde olan öğrencileri ekrana yazdırınız.

3. UYGULAMA

Sözlük Tabanlı Stok Asistanı Geliştiriyorum

Süre: 40 dakika

Amaç: Bir marketteki ürünlerin stok durumunu kontrol eden uygulama geliştirmek

Yönerge: Aşağıdaki adımları takip ederek uygulamayı geliştiriniz.

1. Adım: Editörünüzde **market_yonetim.py** adında yeni bir dosya oluşturunuz. Problemi parçalarına ayırarak bir market stok sisteminin hangi bilgileri tutması ve hangi işlemleri gerçekleştirmesi gerektiğini belirleyiniz.

2. Adım: Marketin stoğunu bir **sözlük** olarak tanımlayınız. Anahtarlar ürün adlarını, değerler ise stok adetlerini tutsun.

```
stok = {"ekmek": 50, "süt": 30, "yumurta": 100, "peynir": 20}
```

3. Adım: Stoğu düzenli biçimde listeleyen bir bölüm yazınız. **items()** metodu ve for döngüsü ile ürün ve adedini ekrana yazdırınız.

4. Adım: Yeni ürün ekleme işlemini yapınız. Kullanıcıdan ürün adını ve stok miktarını alarak stoğa yeni bir anahtar-değer çifti eklemesini isteyiniz.

```
urun = input("Eklenecek ürün: ")
adet = int(input("Adet: "))
stok[urun] = adet
print(urun, "stoğa eklendi.")
```

5. Adım: Ürün çıkartma (satış) işlemini yapınız. Bir ürün satıldığında stok adedini azaltınız, adet sıfıra düştüğünde ürünü **pop()** ile stoktan tamamen çıkarınız.

6. Adım: Stok kontrolü işlemini yapınız. Kullanıcıdan bir ürün adı alınız ve in operatörü ile bu ürünün stokta olup olmadığını kontrol ediniz. Ürün stokta varsa adedini de gösteriniz.

```
aranan = input("Sorgulanacak ürün: ")
if aranan in stok:
    print(aranan, "stokta var. Adet:", stok[aranan])
else:
    print(aranan, "stokta yok.")
```

7. Adım: Programı bir **menü** yapısı içinde toplayınız. Kullanıcı; listeleme, ekleme, satış ve sorgulama işlemlerinden birini seçebilmelidir. Program, kullanıcı çıkış seçeneğini seçene kadar bir döngü içinde çalışmalıdır.

8. Adım: Tamamlanan programınızı README dosyasıyla birlikte dijital deponuza yükleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

SIRA SİZDE

1. Stok adında bir sözlük veri yapısı oluşturunuz.

Sözlüğü for döngüsü ile gezerek adedi 25'in altına düşen ürünleri **STOK AZ** uyarısıyla birlikte ekrana yazdırınız.

Toplam stok adedini (tüm ürünlerin adetleri toplamı) hesaplayınız ve ekranda gösteriniz.

2. Bir kütüphanenin kitap envanterini sözlükle yöneten bir program tasarlayınız. Sözlükte anahtarlar kitap adlarını, değerler ise o kitaptan kaç adet bulunduğunu saklamalıdır.

a) En az dört kitaptan oluşan bir envanter sözlüğü oluşturunuz.

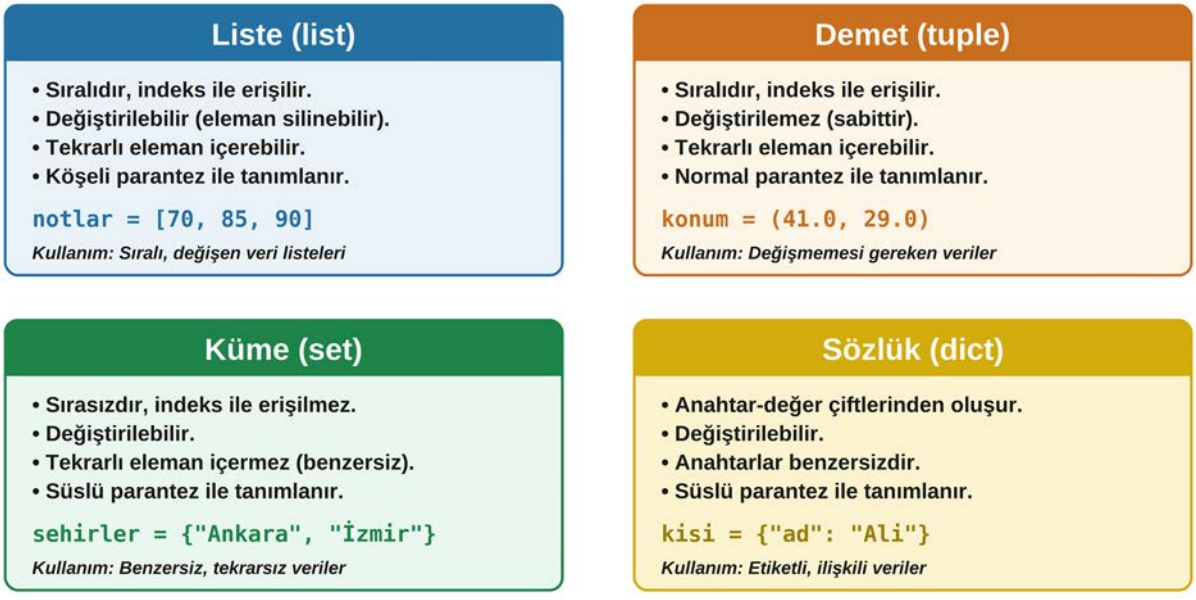
b) Bir kitap ödünç verildiğinde adedini bir azaltan bir bölüm yazınız.

c) Adedi 0 olan kitapları "Tükendi" olarak işaretleyen bir döngü yazınız.

ç) Programınızı farklı senaryolarla test ediniz.

Veri Yapılarının Karşılaştırılması

Bir programcı için en önemli becerilerden biri, hangi durumda hangi veri yapısını kullanacağını bilmesidir. Dört veri yapısının özellikleri Görsel 4.8'de karşılaştırmalı olarak verilmiştir.



Görsel 4.8: Python'da dört temel veri koleksiyonunun karşılaştırması

Doğru veri yapısının seçilmesi, kodun hız ve okunabilirliğini doğrudan etkiler. Verilerin sırası önemliyse ve değiştirilebiliyorsa liste; veriler sabit kalacaksa demet, her eleman benzersiz olmalıysa küme, veriler bir etiketle (anahtarla) saklanacaksa sözlük tercih edilir. Hangi durumda hangi veri yapısının tercih edilmesi gerektiğine ilişkin bilgiler Tablo 4.6'da verilmiştir.

Tablo 4.6: Kullanım Durumlarına Göre Veri Yapılarının Karşılaştırılması

Durum	Uygun Veri Yapısı	Gerekçe
Bir sınıfın sınav notları	Liste	Sıralı ve değiştirilebilir veriler
Bir şehrin koordinatları	Demet	Değişmemesi gereken sabit veriler
Bir yarışmaya katılanlar	Küme	Her katılımcı yalnızca bir kez
Bir öğrencinin kişisel bilgileri	Sözlük	Etiketli (anahtarlı) veriler
Haftanın günleri	Demet	Sabit ve değişmeyen veriler
Ürün adı-fiyat eşleşmesi	Sözlük	Anahtar-değer ilişkisine sahip veriler

47. ÖRNEK UYGULAMA

Amaç: Aynı tipteki verileri farklı veri yapılarıyla saklamak

Bir grup öğrenci adı dört farklı veri yapısında saklanarak her bir veri yapısının içerisindeki bilgiler listelenir.

```
# Liste: sıralı, tekrara izin verir
liste = ["Ali", "Ayşe", "Ali", "Can"]

# Demet: sıralı, değiştirilemez
demet = ("Ali", "Ayşe", "Can")

# Küme: sırasız, tekrarsız
kume = {"Ali", "Ayşe", "Ali", "Can"}

# Sözlük: anahtar–değer ilişkisi
sozluk = {"Ali": 70, "Ayşe": 85, "Can": 90}

print("Liste:", liste)
print("Demet:", demet)
print("Küme:", kume)
print("Sözlük:", sozluk)
```

EKRAN ÇIKTISI

Liste: [‘Ali’, ‘Ayşe’, ‘Ali’, ‘Can’]

Demet: (‘Ali’, ‘Ayşe’, ‘Can’)

Küme: {‘Can’, ‘Ayşe’, ‘Ali’}

Sözlük: {‘Ali’: 70, ‘Ayşe’: 85, ‘Can’: 90}

Liste veri tipi “Ali” adını **iki kez** tutarken küme veri tipi tekrarı eler. Sözlükse her ada bir **not değeri** bağlar. Aynı veri, ihtiyaca göre farklı veri yapılarında saklanır.

SIRA SİZDE

Aşağıdaki her durum için hangi veri yapısının (liste, demet, küme, sözlük) en uygun olduğunu gerekçesiyle birlikte belirleyiniz.

1. Bir kişinin ziyaret ettiği farklı ülkeler (Her ülke bir kez sayılmalı.)
2. Bir öğrencinin haftalık ders programı (gün-ders eşleşmesi)
3. Bir maratonun bitiş sıralaması (Sıra önemlidir ve değişebilir.)
4. Bir ülkenin değişmeyen bayrak renkleri

Veri Koleksiyonlarını Birlikte Kullanmak (İç İçe Yapılar)

Gerçek hayattaki yazılım projelerinde veriler genellikle tek bir liste veya sözlük ile ifade edilemeyecek kadar detaylıdır. Bu durumlarda veri yapıları iç içe kullanılır. Örneğin; bir sözlüğün değerleri listelerden oluşabilir veya bir listenin içinde birden fazla sözlük saklanabilir. Farklı veri yapılarını bir araya getirmek, karmaşık verileri düzenli bir hiyerarşi içinde yönetmeyi sağlar.

48. ÖRNEK UYGULAMA

Amaç: Farklı veri koleksiyonlarını iç içe kullanarak karmaşık verileri yönetmek

Bir şirketteki çalışanların bilgileriyle bildikleri yabancı dilleri bir sözlük içinde saklar. Bu senaryoda sözlüğün anahtarı metin, değeri ise liste olacaktır.

```
# Çalışanlar ve bildikleri diller (Sözlük içinde Listeler)
calisanlar = {
    "Ali": ["İngilizce", "Almanca"],
    "Ayşe": ["İngilizce", "Fransızca", "İspanyolca"],
    "Can": ["Rusça"]
}

# İç içe yapılarda döngü kullanımı
for kisi, diller in calisanlar.items():
    print(kisi, len(diller), "farklı dil biliyor. Diller:", diller)

# Sadece Ayşe'nin bildiği 2. dili (Fransızca) yazdırmak için:
print("Ayşe'nin bildiği 2. dil:", calisanlar["Ayşe"][1])
```

EKRAN ÇIKTISI

Ali 2 farklı dil biliyor. Diller: [‘İngilizce’, ‘Almanca’]

Ayşe 3 farklı dil biliyor. Diller: [‘İngilizce’, ‘Fransızca’, ‘İspanyolca’]

Can 1 farklı dil biliyor. Diller: [‘Rusça’]

Ayşe’nin bildiği 2. dil: Fransızca

BİLGİ KUTUSU

Python’daki özellikle sözlük ve liste kombinasyonları gibi iç içe veri koleksiyonları, modern yazılım geliştirme ve web programlamada veri alışverişi için standart olarak kullanılan **JSON (JavaScript Object Notation)** veri formatının temelini oluşturur. Bir mobil uygulamanın hava durumu bilgisini veya bir e-ticaret sitesindeki ürün listesini çekerken arka planda her zaman bu iç içe koleksiyon mimarisi çalışır.

4.1. PROJE	
Konu	Kütüphane Yönetim Sistemi Yazıyorum
Kapsanan Konular	Veri koleksiyonları, liste, demet, küme sözlük
Süre	40+40+40 dakika
Amaç	Veri koleksiyonlarını kapsayıcı bir projede kullanmak
Çalışma Biçimi	Bireysel proje

Proje Senaryosu

Okulunuzdaki kütüphaneyi yönetmek için bir program geliştiriniz. Bu programda kitapların bilgilerini (kitap adı, yazar adı, yayın yılı) bir demet olarak yazarak kütüphanedeki tüm kitapları bir liste içinde tutunuz. Üyelerin ödünç aldığı kitapları bir sözlük şeklinde kaydediniz ve mevcut yazarları bir küme olarak düzenleyiniz. Bu sayede dört veri yapısının tamamını anlamlı bir şekilde kullanarak kütüphanedeki bilgileri düzenleyiniz.

Yönerge: Aşağıdaki adımları takip ederek projeyi geliştiriniz.

1. Adım: Planlama: Tüm projenin algoritmasını yazınız. Hangi işlemde, hangi veri yapısının kullanılacağına önceden karar veriniz.

2. Adım: Geliştirme Ortamı: Editörünüzde **kutuphane_yonetim.py** dosyası oluşturunuz.

3. Adım: Demet-Kitap Bilgileri: Her kitabı bir demet olarak tutunuz. Demet açma ile bilgilere erişiniz.

```
# (kitap_adi, yazar, yayin_yili)
kitap1 = (“Kürk Mantolu Madonna”, “Sabahattin Ali”, 1943)
kitap2 = (“Çalıkuşu”, “Reşat Nuri Güntekin”, 1922)
```

4. Adım: Liste-Kütüphane: Tüm kitapları bir liste içinde tutunuz. Demetler bu listenin elemanlarıdır.

```
kutuphane = [
    (“Kürk Mantolu Madonna”, “Sabahattin Ali”, 1943),
    (“Çalıkuşu”, “Reşat Nuri Güntekin”, 1922),
    (“İnce Memed”, “Yaşar Kemal”, 1955),
    # ... en az 8 kitap olsun
]
```

5. Adım: Sözlük-Ödünç Verilenler: Kitap-üye eşleşmesi için bir sözlük tutunuz. Anahtar kitap adı; değer ise üye adıdır.

```
odunc = {
    “Çalıkuşu”: “A*** Y*****”,
    # ...
}
```

6. Adım: Küme-Mevcut Yazarlar: Kütüphanedeki **tüm yazarları** bir küme olarak tutunuz (tekrarsız). Liste içinden yazarları çıkarmak için bir döngü yazınız.

7. Adım: Menü (While ve if/elif): Aşağıdaki seçenekleri içeren bir menü oluşturunuz. Kullanıcı çıkış seçeneğini seçmeden program devam etsin.

```
print("\n1) Tüm kitapları göster")
print("2) Kitap ara (kitap adına göre)")
print("3) Yazara göre kitapları göster")
print("4) Kitap ödünç ver")
print("5) Kitap iade al")
print("6) Ödünç verilenleri göster")
print("7) Mevcut yazar listesi")
print("8) Yeni kitap ekle")
print("9) Çıkış")
```

8. Adım: Tüm kitapları göster. Demet açma ile kitapları formatlı yazdırınız.

9. Adım: Kitap ara. in operatörü ve bir for döngüsüyle kitap adını arayınız.

10. Adım: Yazara göre kitap göster: Kullanıcıdan yazar adı alıp onun yazdığı tüm kitapları gösteriniz.

11. Adım: Kitap ödünç ver. Kitabın kütüphanede olup olmadığını kontrol ediniz. Kitap ödünç verilmişse uyarı veriniz değilse sözlüğe ekleyiniz.

12. Adım: Kitap iade al. Sözlükten kitabı siliniz ama iade edilmeden önce sözlükte olup olmadığını kontrol ediniz.

13. Adım: Mevcut yazar listesi: Yazarlar kümesini ekrana yazdırınız.

14. Adım: Yeni kitap ekle. Kullanıcıdan ad, yazar adı, yıl bilgilerini alıp listeye demet olarak ekleyiniz. Yazarlar kümesini de güncelleyiniz.

15. Adım: Test: Programınızı beş-altı farklı işlem senaryosuyla test ediniz: kitap ekleme, arama, ödünç verme, iade vb.

16. Adım: Dijital Depo ve Sürüm (Release): Projenizi dijital deponuza yükleyiniz. README.md dosyasında projenin amacını ve nasıl kullanılacağını yazınız. Çalışmanızın çalışan ilk tam hâlini temsil etmesi için deponuzda **v1.0 adıyla bir Release (sürüm) oluşturunuz.** Bu süreç boyunca anlamlı commit mesajları yazmayı unutmayınız.

Değerlendirme:

a) Yapacağınız proje çalışması öğretmeniniz tarafından **Proje Derecelendirme Ölçeğinde** yer alan ölçütlere göre değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak derecelendirme ölçeğine ulaşabilirsiniz.

Proje derecelendirme ölçeğine ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

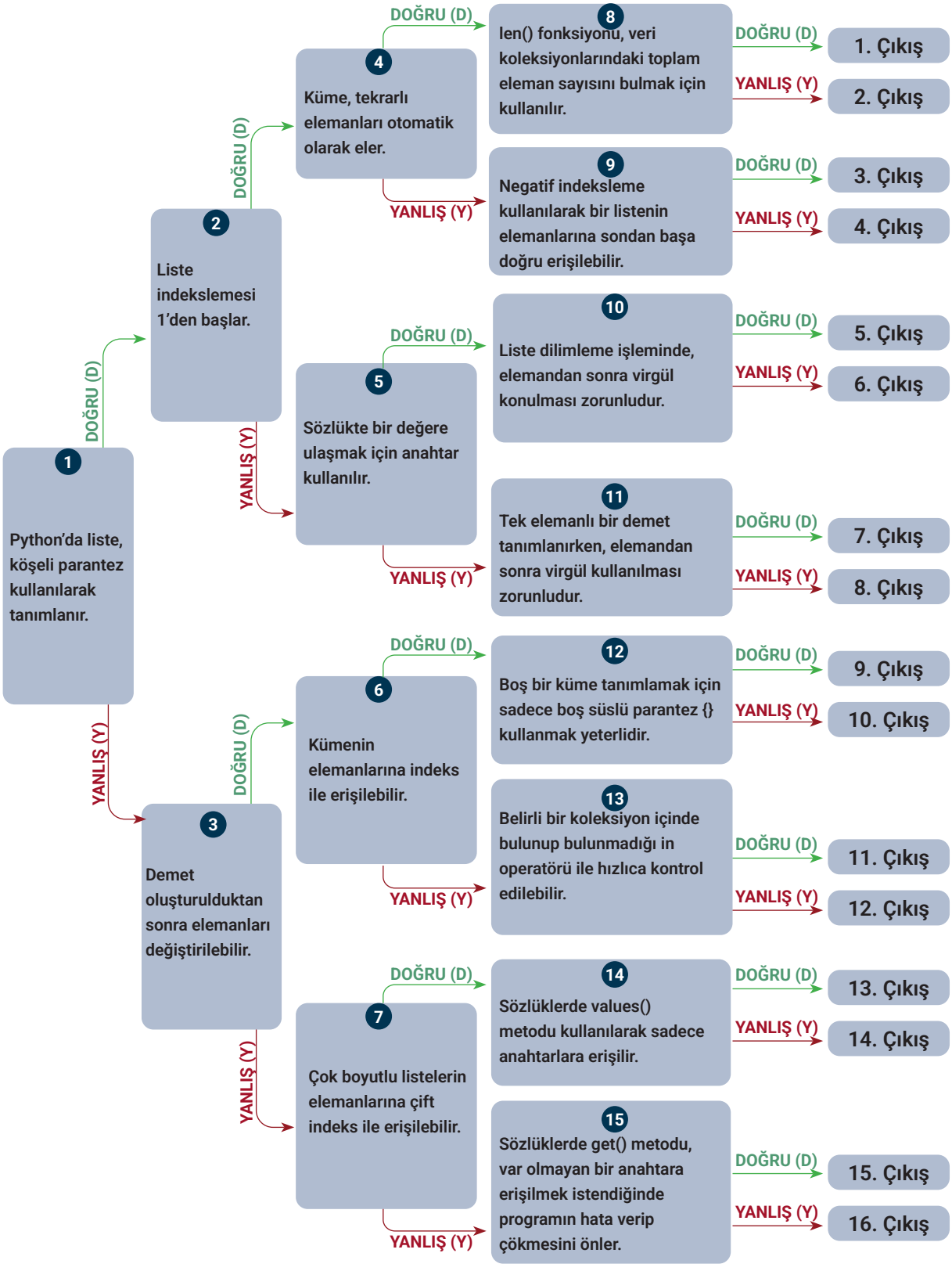
b) **Öz Değerlendirme Formu** proje geliştirme sürecinin sonunda kendinizi değerlendirmeniz amacıyla hazırlanmıştır. Cevaplarınızı tabloda ilgili kutucuğa yazınız. Verilen karekodu kullanarak öz değerlendirme formuna ulaşabilirsiniz.

Öz değerlendirme formuna ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

ÖLÇME VE DEĞERLENDİRME

A) Aşağıda Doğru / Yanlış önermeleri içeren tanılayıcı dallanmış ağaç verilmiştir. Verilen önermenin doğru olduğunu düşünüyorsanız (D), yanlış olduğunu düşünüyorsanız (Y) seçeneğini işaretleyiniz. İlk önermeden başlayarak yapacağınız seçimlerle ve oklar yönünde ilerleyerek bir çıkış noktası belirleyiniz.



B) Aşağıdaki cümlelerde boş bırakılan yerleri doğru kelimelerle doldurunuz.

1. Birden fazla veriyi tek bir değişken altında sıralı biçimde saklayan veri yapısına ..... denir.

2. Bir listenin elemanına ulaşmak için kullanılan sıra numarasına ..... denir.

3. Bir listenin belirli bir aralığını almak için kullanılan işleme ..... denir.

4. Listenin sonuna eleman eklemek için ..... metodu kullanılır.

5. Oluşturulduktan sonra değiştirilemeyen veri yapısına ..... denir.

6. Tekrarsız ve sırasız elemanlardan oluşan veri yapısına ..... denir.

7. Verileri anahtar-değer çiftleri biçiminde saklayan veri yapısına ..... denir.

8. İçinde liste bulunan listelere ..... liste veya matris denir.

9. Bir sözlükte var olmayan bir anahtara güvenli biçimde erişmek için ..... metodu kullanılır.

10. İki kümenin ortak elemanlarını bulan işleme ..... denir.

C) Sol sütundaki veri yapısını, sağ sütundaki doğru özelliğe ait harfle eşleştiriniz.

	Veri Yapısı
( )	1. Liste
( )	2. Demet
( )	3. Küme
( )	4. Sözlük

Özellik
a) Anahtar-değer çiftlerinden oluşur.
b) Sıralı ve değiştirilebilir
c) Değiştirilemez (sabit)
ç) Tekrarsız ve sırasız
d) Tek bilgi atanabilir.

Ç) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Aşağıdakilerden hangisi bir liste tanımıdır?

A) [1, 2, 3]
B) {1, 2, 3}
C) (1, 2, 3)
D) {“a”: 1}
E) 1, 2, 3

2. meyveler = [“elma”, “armut”, “kiraz”] listesinde meyveler[-1] ifadesi hangi değeri verir?

A) -1
B) Hata oluşur
C) elma
D) armut
E) kiraz

3.

```
sayilar = [10, 20, 30, 40, 50]

print(sayilar[1:3])
```

Yukarıdaki kodun çıktısı nedir?

A) [10, 20]
B) [20, 30]
C) [30, 40]
D) [10, 20, 30]
E) [20, 30, 40]

4.

```
sayilar = {1, 2, 2, 3, 3, 3}

print(len(sayilar))
```

Yukarıdaki kodun çıktısı nedir?

A) 1
B) 2
C) 3
D) 6
E) 9

5. Bir kümeye eleman eklemek için aşağıdaki metotlardan hangisi kullanılır?

A) add()
B) append()
C) insert()
D) push()
E) update()

6. ogrenci = {“ad”: “Ali”, “yas”: 16} sözlüğünde ogrenci[“yas”] ifadesi hangi değeri verir?

A) ad
B) Ali
C) 16
D) yas
E) Hata

7. Bir listedeki tekrarlı elemanları temizlemenin en pratik yolu aşağıdakilerden hangisidir?

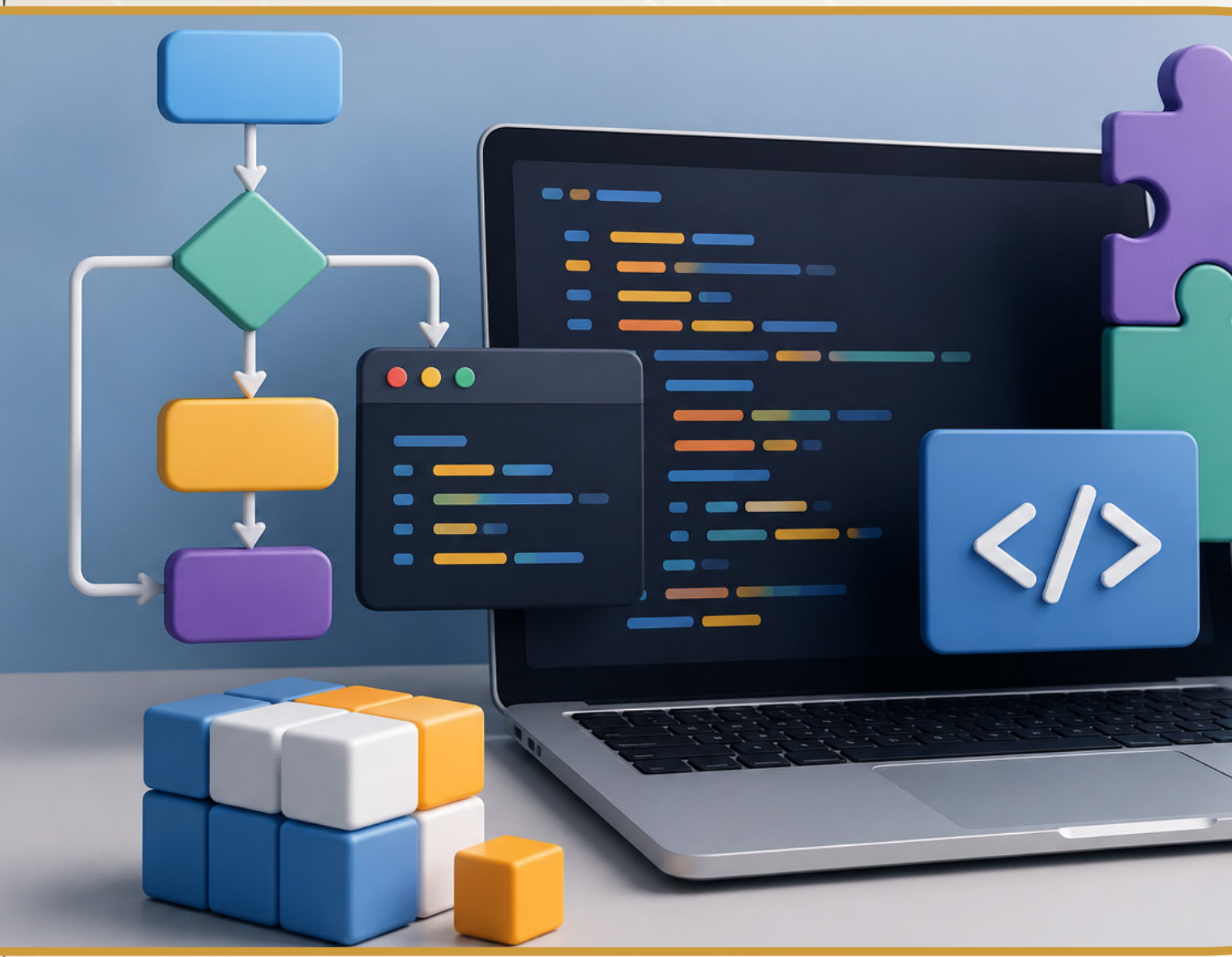
A) Listeyi demete çevirmek
B) Listeyi kümeye çevirmek
C) Listeyi sözlüğe çevirmek
D) sort() metodu kullanmak
E) reverse() metodu kullanmak

8. Değişmemesi gereken sabit veriler için en uygun veri yapısı hangisidir?

A) Liste
B) Matris
C) Küme
D) Sözlük
E) Demet

D) Bu öğrenme biriminde öğrendiklerinizi değerlendirmek için aşağıdaki çıkış kartını doldurunuz.

Üç Yaz	**Veri Koleksiyonları öğrenme biriminde öğrendiğiniz üç bilgiyi yazınız.**
1	
2	
3	
İki Sor	**Veri yapıları ile ilgili merak ettiğiniz iki soruyu yazınız.**
1	
2	
Bir Paylaş	**Bu öğrenme biriminde geliştirdiğiniz uygulamalardan birini sınıf arkadaşlarınızla paylaşmak isteseniz hangisini seçerdiniz? Neden?**



KONULAR

5.1. FONKSİYONLAR

5.2. MODÜLER PROGRAMLAMA

ANAHTAR KAVRAMLAR

- değer döndürme
- fonksiyon
- kapsam
- lambda fonksiyonu
- modüler programlama
- özyineleme
- parametre
- yerel değişken



5. ÖĞRENME BİRİMİ

FONKSİYONLAR VE MODÜLER PROGRAMLAMA

BEKLENTİLER

Bu öğrenme biriminde sizden

- fonksiyonları tanımlayarak modüler programlar yazmanız
- modüler mimariyi kullanarak programlar yazmanız

beklenmektedir.

HAZIRLIK ÇALIŞMALARI

1. Aşağıda bir fabrikanın üretim hattı ile ilgili bir metin verilmiştir.

"Bir otomobil fabrikasında her araç baştan sona bir kişi tarafından üretilmez. Üretim hattı; motor montajı, boyama, iç döşeme, kalite kontrol gibi farklı istasyonlara ayrılmıştır. Her istasyon kendi görevini yerine getirir ve işlemini tamamladıktan sonra aracı bir sonraki istasyona gönderir. Herhangi bir istasyonda sorun çıktığında tüm fabrika durdurulmaz yalnızca ilgili istasyon incelenir. Boyama istasyonu bir kez kurulduğunda her araç için yeniden kurulmasına gerek kalmadan tekrar kullanılabilir."

Verilen metne göre aşağıdaki soruları cevaplayınız.

a) Üretimin istasyonlara bölünmesi, bir sorun çıktığında neden avantajlıdır?

b) Üretim istasyonunda olduğu gibi bir bilgisayar programını küçük ve bağımsız parçalara bölmek programcıya hangi kolaylıkları sağlar?

2. Daha önce yazdığınız programlarda aynı işi yapan kod bloklarını tekrar tekrar yazdınız. Sizce bu durum hangi zorluklara neden olur?

BAŞLARKEN

Büyük ve karmaşık işler **küçük**, **bağımsız parçalara** bölündüğünde yönetim kolaylaşır hatalarda daha hızlı bulunur. Programlamada da büyük bir problemi tek bir bütün hâlinde çözmek yerine onu yönetilebilir küçük parçalara ayırmak temel bir yaklaşımdır.

Yazılan programlar büyüdükçe, aynı işi yapan kod blokları birkaç defa yazılmış olabilir. Bu durum zaman kaybına neden olurken hata olasılığını artırır. **Fonksiyonlar**, bir kez yazılıp defalarca kullanılabilen kod bloklarıdır ve modüler programlamanın temel aracıdır.

5.1. FONKSİYONLAR

Program genişledikçe içindeki kod satırları da genişler. Yüzlerce hatta binlerce satırlık programı tek bir bütün hâlinde yazmak ve yönetmek son derece zordur. Aynı işi yapan kod parçaları programın farklı yerlerinde tekrar kullanılmak istenebilir. Bu amaçla fonksiyonlar kullanılır.

Fonksiyon Kavramı

Belirli bir işi yapmak üzere bir araya getirilmiş, bir ada sahip ve gerektiğinde tekrar tekrar çağrılabilen kod bloğuna **fonksiyon** denir. Fonksiyonlar, büyük bir programın küçük ve yönetilebilir parçalara bölünmesini sağlar. Bu yaklaşıma **modüler programlama** denir.

Günlük hayatta da fonksiyona benzeyen yapılarla karşılaşılır. Hesap makinesindeki **karekök** tuşuna basıldığında sadece bir sayı girilerek karekökü alınabilir. Tuş bir fonksiyonu çalıştırır, girdi alır, işlem yapar ve sonuç verir.

Fonksiyon kullanmanın başlıca yararları şunlardır:

- **Kod tekrarını önler:** Aynı işi yapan kod bir kez yazılır ve defalarca kullanılır.
- **Programı düzenler:** Büyük problemler, küçük ve anlaşılır parçalara bölünür.
- **Bakımı kolaylaştırır:** Bir hata olduğunda yalnızca ilgili fonksiyon incelenir.
- **Okunabilirliği artırır:** Anlamlı adlara sahip fonksiyonlar, kodun ne yaptığını açıkça gösterir.

BİLGİ KUTUSU

Python'da bazı fonksiyonlar hazır olarak gelir, bunlara yerleşik (built-in) fonksiyonlar adı verilir. print(), input(), int(), len() bu fonksiyonlara örnektir.

Bir programın üç farklı bölümünde, öğrenciye ait not ortalamalarının hesaplanması gerektiğini düşününüz. Program fonksiyon kullanılmadan yazılırsa ortalama hesaplama kodu üç kez yazılmak zorunda kalır. Ortalama hesaplama kod mantığında değişiklik yapılması gerektiğinde ise her üç bölümdeki kodun tekrar değiştirilmesi gerekir. Kodları tekrar yazmak zaman alıcıdır. Ayrıca kod bölümünde ilgili değişiklik unutulursa karışıklığa neden olur. Aynı işlem bir ortalama_al fonksiyonu ile yazılırsa kod bir kez yazılır ve yalnızca o fonksiyon düzeltilerek tüm program güncellenmiş olur.

Fonksiyonların Günlük Hayattaki Karşılığı

Tablo 5.1'de bazı günlük hayat örnekleri fonksiyon kavramıyla ilişkilendirilmiştir.

Tablo 5.1: Fonksiyonlar İçin Günlük Hayat Örnekleri

Günlük Hayat Örneği	Girdi	İşlem	Çıktı
Çamaşır makinesi	Kirli çamaşır ve deterjan	Yıkama programı	Temiz çamaşır
Kahve makinesi	Su, kahve çekirdeği	Demleme	Bir fincan kahve
Hesap makinesi (karekök)	Bir sayı	Karekök işlemi	Sayının karekökü
Para çekme makinesi	Kart, şifre, tutar	Hesap kontrolü	Para ve fiş

Her örnek, girdi alarak belirli bir işlem yapar ve bir çıktı üretir. Fonksiyonlar da parametre olarak girdi alır, gövdesinde bir işlem yapar ve return ifadesiyle çıktı üreterek çalışır.

Modüler Programlama Yaklaşımı

Fonksiyonlar, **modüler programlama** yaklaşımının temel aracıdır. Modüler programlama; büyük bir problemi küçük, bağımsız ve yönetilebilir parçalara (modüllere) bölme yaklaşımıdır. Her modül belirli bir işi yapar ve diğer modüllerle birlikte çalışarak bütünü oluşturur.

Bu yaklaşım bir ev inşaatına benzetilebilir. Bir evin yapımında temel, duvar, çatı, elektrik tesisatı, su tesisatı gibi ayrı işler vardır ve bu işler farklı ekipler tarafından yapılır. Elektrik arızası gibi bir sorun yaşandığında tüm evi yıkmak yerine yalnızca elektrik tesisatı incelenir. Aynı şekilde modüler bir programda da bir hata olduğunda yalnızca ilgili fonksiyon incelenir.

Modüler programlama kullanıcıya birçok avantaj sağlar. Programın anlaşılması kolaylaşır, hata bulma süreci hızlanır, kod parçaları başka projelerde tekrar kullanılabilir. Birden fazla programcı aynı proje üzerinde farklı fonksiyonlar yazarak birlikte çalışabilir.

BİLGİ KUTUSU

Profesyonel yazılım projeleri binlerce hatta milyonlarca satır koddan oluşur. Bu projeler, yüzlerce küçük fonksiyonun bir araya gelmesiyle yönetilir. Modüler programlama olmadan bu büyüklükte projeler geliştirmek neredeyse imkânsızdır. Bu nedenle fonksiyonlar, modern programlamanın temel taşıdır.

1. ÖRNEK UYGULAMA

Amaç: Aynı programlama işini fonksiyon kullanmadan ve fonksiyon kullanarak yazmak

Bir öğrencinin üç farklı derse ait not ortalamasını hesaplayan program önce fonksiyon kullanılmadan ardından fonksiyon kullanılarak yazılması amaçlanır. Fonksiyon kullanılmadan oluşturulan kod aşağıda şekilde gösterilir:

```
1  # Fonksiyonsuz yazım – kod tekrar ediyor
2  mat_ort = (70 + 80 + 90) / 3
3  print("Matematik ortalaması:", mat_ort)
4
5  fiz_ort = (60 + 65 + 70) / 3
6  print("Fizik ortalaması:", fiz_ort)
7
8  kim_ort = (85 + 90 + 95) / 3
9  print("Kimya ortalaması:", kim_ort)
```

EKRAN ÇIKTISI
Matematik ortalaması: 80.0
Fizik ortalaması: 65.0
Kimya ortalaması: 90.0

Aynı program, bir fonksiyon kullanılarak şöyle yazılır:

```
1  # Fonksiyonlu yazım – kod tekrarı yok
2  def ortalama(n1, n2, n3):
3      return (n1 + n2 + n3) / 3
4
5  print("Matematik ortalaması:", ortalama(70, 80, 90))
6  print("Fizik ortalaması:", ortalama(60, 65, 70))
7  print("Kimya ortalaması:", ortalama(85, 90, 95))
```

EKRAN ÇIKTISI
Matematik ortalaması: 80.0
Fizik ortalaması: 65.0
Kimya ortalaması: 90.0

İki yazım da görüldüğü gibi aynı sonucu verir ancak fonksiyonlu yazımda ortalama hesaplama kodu **yalnızca bir kez** yazılır. Ortalama hesaplama biçimi değişirse (örneğin ağırlıklı ortalamaya geçilirse) fonksiyonlu yazımda yalnızca tek bir yeri düzeltmek yeterlidir.

SIRA SİZDE

1. Günlük hayatınızdan girdi, işlem ve çıktısı olan ve fonksiyon şeklinde yazılabilecek bir örnek yazınız.

2. Örnek fonksiyonun üç temel ögesini (girdi, işlem, çıktı) belirleyerek yazınız.

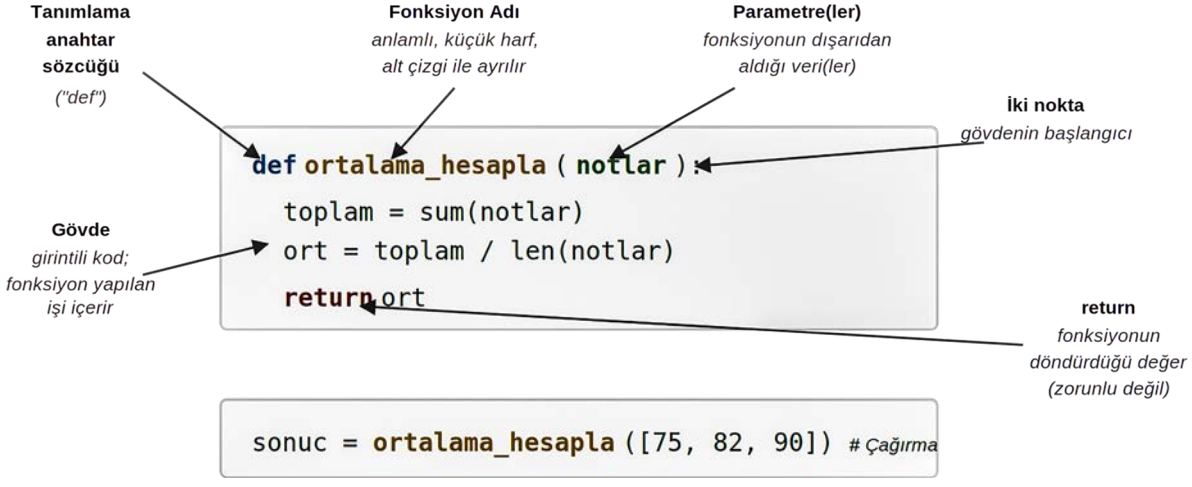
3. Yazdığınız örneği fonksiyon olarak kodlayınız.

Fonksiyon Tanımlama ve Çağırma

Python'da bir fonksiyon def (define = tanımlamak) anahtar sözcüğü ile tanımlanır. Fonksiyon tanımının genel söz dizimi şu şekildedir:

```
def fonksiyon_adi():
    # Fonksiyonun yapacağı işler (gövde)
    yapilacak_islemler
```

Bir fonksiyon yalnızca **tanımlandığında** çalışmaz, çalışması için **çağrılması** gerekir. Fonksiyon, adının yanına parantez yazılarak çağrılır. Görsel 5.1'de fonksiyon yapısının bileşenleri bir örnek üzerinden gösterilir.



Görsel 5.1: Bir fonksiyonun parçaları

2. ÖRNEK UYGULAMA

Amaç: Ekrana sabit bir mesaj yazdıran fonksiyon yazmak

```
# Fonksiyon TANIMI
def selam_ver():
    #Ekrana selam mesajı yazdırır.
    print("Merhaba, hoş geldiniz!")
    print("Programa başlanıyor...")
# Fonksiyon ÇAĞRISI
selam_ver()    # Mesajları yazdırır
selam_ver()    # Tekrar çağrılırsa tekrar yazar
selam_ver()    # İstediğiniz kadar çağırabilirsiniz
```

EKRAN ÇIKTISI

```
Merhaba, hoş geldiniz!
Programa başlanıyor...
Merhaba, hoş geldiniz!
Programa başlanıyor...
Merhaba, hoş geldiniz!
Programa başlanıyor...
```

Buradaki örnekte **selam_ver** adında bir fonksiyon tanımlandı. Fonksiyon, **selam_ver** yazılarak çağrıldığında gövdesindeki iki print satırı çalıştırıldı.

3. ÖRNEK UYGULAMA

Amaç: Bir fonksiyonu birden fazla kez çağırmak

Fonksiyonların en büyük avantajı, bir kez tanımlanıp defalarca çağrılabilmeleridir.

```
def cizgi_ciz():
    print("-----------------------")

print("Rapor Başlığı")
cizgi_ciz()
print("İçerik buraya gelecek")
cizgi_ciz()
print("Rapor Sonu")
```

EKRAN ÇIKTISI

```
Rapor Başlığı
-----------------------
İçerik buraya gelecek
-----------------------
Rapor Sonu
-----------------------
```

cizgi_ciz fonksiyonu bir kez tanımlandı ama üç kez çağrıldı. Çizginin biçimi değiştirilmek istenseydi yalnızca fonksiyon tanımını değiştirmek yeterli olurdu ve tüm çağrılar otomatik olarak güncellenirdi.

4. ÖRNEK UYGULAMA

Amaç: Aynı fonksiyonu farklı görevlerde kullanmak

Aşağıdaki örnekte, menü ekranı oluşturulurken aynı yardımcı fonksiyonun nasıl kullanıldığı gösterilir.

```
1  def bosluk_birak():
2      print()
3
4  print("ANA MENÜ")
5  bosluk_birak()
6  print("1 - Yeni Kayıt")
7  print("2 - Kayıt Listele")
8  print("3 - Çıkış")
9  bosluk_birak()
10 print("Seçiminizi yapınız.")
```

EKRAN ÇIKTISI

```
ANA MENÜ


1 - Yeni Kayıt
2 - Kayıt Listele
3 - Çıkış


Seçiminizi yapınız.
```

Basit bir iş olan boş satır bırakmak, fonksiyon hâline getirildiğinde kodun okunabilirliği artırılır. bosluk_birak() yazıldığında neden boş satır bırakıldığı da anlaşılır.

BİLGİ KUTUSU

Fonksiyon, çağrılmadan önce tanımlanmış olmalıdır. Python kodu yukarıdan aşağıya çalıştırdığı için henüz tanımlanmamış fonksiyonu çağırmak **NameError** hatasına yol açar. Bu nedenle fonksiyon tanımları genellikle programın başında yer alır.

5. ÖRNEK UYGULAMA

Amaç: Bir fonksiyonun gövdesinde başka bir fonksiyonu çağırmak

Bir fonksiyonun gövdesinde başka bir fonksiyon çağırmak modüler programlamanın temel ilkelerinden biri olarak gösterilir.

```
1  def baslik_yaz():
2      print("=== ÖĞRENCİ BİLGİ SİSTEMİ ===")
3
4  def alt_bilgi_yaz():
5      print("Tüm hakları saklıdır.")
6
7  def ekran_olustur():
8      baslik_yaz()
9      print("İçerik burada gösterilir.")
10     alt_bilgi_yaz()
11
12 # Sadece ana fonksiyon çağrılıyor
13 ekran_olustur()
```

EKRAN ÇIKTISI

```
=== ÖĞRENCİ BİLGİ SİSTEMİ ===
İçerik burada gösterilir.
Tüm hakları saklıdır.
```

Burada **ekran_olustur** fonksiyonu tarafından **baslik_yaz** ve **alt_bilgi_yaz** fonksiyonları çağırılır. Büyük bir iş, küçük fonksiyonların birlikte çalışmasıyla tamamlanır. Bu sayede program düzenli ve anlaşılır tutulur. Burada dikkat edilmesi gereken **ekran_olustur** fonksiyonun **baslik_yaz** ve **alt_bilgi_yaz** fonksiyonlarından sonra yazılmasıdır.

SIRA SİZDE

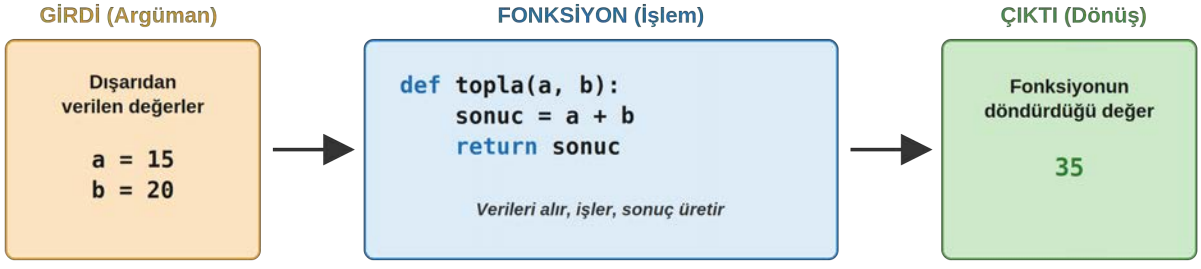
Aşağıdaki işleri yapan fonksiyonları tanımlayarak ana programdan çağırınız.

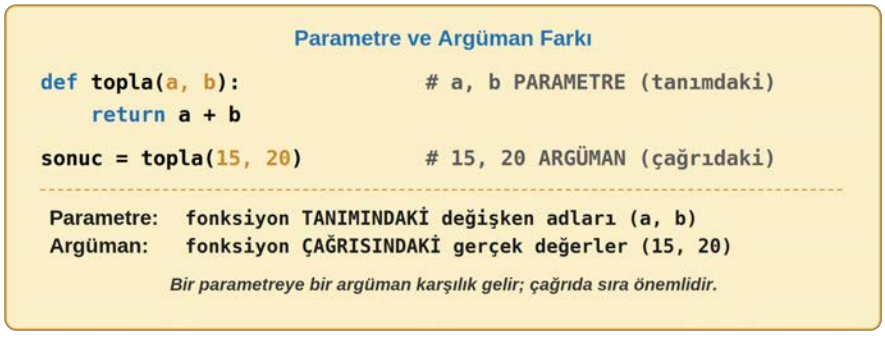
1. Ekrana karşılama mesajı yazan karsila fonksiyonu
2. Ekrana bugünün önemini anlatan bir cümle yazan gun_mesaji fonksiyonu
3. Ekrana veda mesajı yazan vedalas fonksiyonu
4. 1, 2, 3. madde için tanımladığınız fonksiyonları sırayla çağıran program_calistir adında bir fonksiyon yazınız ve ana programdan yalnızca program_calistir fonksiyonunu çağırınız.

Parametre ve Argümanlar

Fonksiyonlar her çağrıldığında aynı işi yapar ancak çoğu zaman fonksiyonların dışarıdan veri alarak çalışması istenir. Fonksiyona dışarıdan veri aktarmak için **parametreler** kullanılır.

Parametre, fonksiyon tanımlanırken parantez içine yazılan değişkendir. **Argüman** fonksiyon çağrılırken parametreye gönderilen gerçek değerdir. Bir fonksiyonun çalışma sürecinde argüman ve parametrenin görevi Görsel 5.2'de gösterilmiştir.





Görsel 5.2: Parametre ve argüman arasındaki fark

6. ÖRNEK UYGULAMA

Amaç: Tek parametreli bir fonksiyon tanımlamak

```
1   # Parametreli fonksiyon TANIMI
2   def selamla(ad):     # ad → PARAMETRE
3       print(f"Merhaba, {ad}!")
4
5   # Fonksiyon ÇAĞRILARI
6   selamla("Ahmet")     # "Ahmet" → ARGÜMAN
7   selamla("Ayşe")
8   selamla("Mehmet")
```

EKRAN ÇIKTISI
Merhaba, Ahmet!
Merhaba, Ayşe!
Merhaba, Mehmet!

Burada **ad** bir parametre olarak tanımlanır. "Ahmet", "Ayşe" ve "Mehmet" her çağrıda gönderilen argümandır. Aynı fonksiyonda farklı argümanlarla farklı sonuçlar üretilir.

7. ÖRNEK UYGULAMA

Amaç: Birden fazla parametreli bir fonksiyon tanımlamak

Bir fonksiyon tarafından birden fazla parametre alınabilir. Parametreler virgülle ayrılır.

```
1   def topla(a, b):
2       sonuc = a + b
3       print(f"{a} + {b} = {sonuc}")
4
5   topla(5, 3)
6   topla(10, 20)
7
8   def selamla(ad, soyad, yas):
9       print(f"Merhaba {ad} {soyad}, {yas} yaşındasınız.")
10
11  selamla("A****", "Y*****", 15)
```

EKRAN ÇIKTISI

```
5 + 3 = 8
10 + 20 = 30
Merhaba A**** Y*****, 15 yaşındasınız.
```

Argümanlar, parametrelere **yazıldıkları sıraya göre** aktarılır. İlk örnekteki topla fonksiyonunda argüman (5) a parametresine, ikinci argüman (3) b parametresine gönderilir. İkinci örnekte **selamla** fonksiyonuna üç parametre gönderilmektedir. ("Ahmet") argümanı **ad** parametresine ("Yılmaz") argümanı **soyad** parametresine ve (15) argümanı da **yas** parametresine gönderilir.

8. ÖRNEK UYGULAMA

Amaç: Parametreli fonksiyonla hesaplama yapmak

Aşağıdaki fonksiyonda, bir ürünün adı ve fiyatı alınarak bir bilgi satırı oluşturulur.

```
1   def urun_etiketi(ad, fiyat, adet):
2       toplam = fiyat * adet
3       print("Ürün:", ad)
4       print("Birim fiyat:", fiyat, "TL")
5       print("Adet:", adet)
6       print("Toplam tutar:", toplam, "TL")
7       print("-" * 25)
8
9   urun_etiketi("Defter", 30, 4)
10  urun_etiketi("Kalem", 15, 10)
```

EKRAN ÇIKTISI

```
Ürün: Defter
Birim fiyat: 30 TL
Adet: 4
Toplam tutar: 120 TL
----------
Ürün: Kalem
Birim fiyat: 15 TL
Adet: 10
Toplam tutar: 150 TL
----------
```

Üç parametreli bu fonksiyonda her çağrıda farklı ürün bilgileriyle çalışarak düzenli bir ürün etiketi üretilir. Aynı işi yapan kodu her ürün için tekrar yazmak yerine fonksiyon bir kez tanımlanır ve defalarca kullanılabilir.

9. ÖRNEK UYGULAMA

Amaç: Varsayılan parametre değeri kullanmak

Bir parametreye varsayılan bir değer atanabilir. Bu durumda fonksiyon çağrılırken o argüman verilmezse varsayılan değer kullanılır.

```
1  def selamla(ad, dil="Türkçe"):
2      if dil == "Türkçe":
3          print(f"Merhaba, {ad}!")
4      elif dil == "İngilizce":
5          print(f"Hello, {ad}!")
6      elif dil == "Almanca":
7          print(f"Hallo, {ad}!")
8
9  selamla("Ahmet")
10 selamla("John", "İngilizce")
11 selamla("Hans", "Almanca")
```

EKRAN ÇIKTISI

```
Merhaba, Ahmet!
Hello, John!
Hallo, Hans!
```

İlk çağrıda dil argümanı verilmediği için varsayılan değer (“Türkçe”) kullanılır. İkinci çağrıda verilen argüman kullanılır.

10. ÖRNEK UYGULAMA

Amaç: Varsayılan parametre ile esnek fonksiyon tasarlamak

Varsayılan parametrelerle bir fonksiyon basit ve ayrıntılı kullanıma uygun hâle getirilir. Aşağıdaki fonksiyonla bir alan hesaplaması yapılır.

```
1  def kdv_hesapla(fiyat, oran=20):
2      kdv = fiyat * oran / 100
3      return fiyat + kdv
4
5  # Varsayılan oran (%20) ile
6  print("Standart KDV'li fiyat:", kdv_hesapla(1000))
7
8  # Farklı oran (%10) ile
9  print("İndirimli KDV'li fiyat:", kdv_hesapla(1000, 10))
```

EKRAN ÇIKTISI
Standart KDV'li fiyat: 1200.0
İndirimli KDV'li fiyat: 1100.0

Fonksiyon tarafından çoğu durumda kullanılan standart KDV oranını (**%20**) varsayılan değer olarak alınır. Farklı bir oran gerektiğinde ikinci argüman verilir. Böylece tek bir fonksiyon hem yaygın hem de özel durumlar için kullanılabilir.

SIRA SİZDE

1. Bir öğrencinin adını ve isteğe bağlı olarak selamlama biçimini parametre olarak alan selam_ver(ad, selam="Merhaba") fonksiyonunu yazınız.

2. Fonksiyonu önce ad vererek sonra ad ve farklı bir selamlama biçimi ("Günaydın" gibi) vererek çağırınız. Varsayılan parametrenin ne zaman devreye girdiğini gözlemleyiniz.

BİLGİ KUTUSU

Parametre ile argüman kavramları sıkça karıştırılır. Parametre fonksiyon tanımlanırken kullanılan değişken argüman ise fonksiyon çağrılırken gönderilen gerçek değerdir. Parametre "kutu", argüman ise "kutuya konan değer" olarak düşünülebilir.

11. ÖRNEK UYGULAMA

Amaç: Parametre olarak farklı veri türleri kullanmak

Fonksiyon parametreleriyle yalnızca sayılar değil metin, liste gibi her tür veri alınabilir.

```
1  def liste_ozeti(isim, notlar):
2      toplam = sum(notlar)
3      ortalama = toplam / len(notlar)
4      print(isim, "öğrencisinin not sayısı:", len(notlar))
5      print(isim, "öğrencisinin ortalaması:", ortalama)
6
7  ogrenci_notlari = [70, 85, 90, 65]
8  liste_ozeti("Zeynep", ogrenci_notlari)
```

EKRAN ÇIKTISI
Zeynep öğrencisinin not sayısı: 4
Zeynep öğrencisinin ortalaması: 77.5

Burada **isim** parametresi metin, **notlar** parametresi ise liste olarak alınır. Fonksiyonlar tarafından farklı veri türleri parametre olarak alabilen ve oldukça esnek olan yapılardır.

12. ÖRNEK UYGULAMA

Amaç: Anahtar kelimeli argüman kullanmak

Bir fonksiyon çağrılırken argümanlar, parametre adı belirtilerek gönderilir. Buna **anahtar kelimeli argüman** adı verilir.

```
1   def bilgi(ad, yas, sehir):
2       print(f"{ad}, {yas} yaşında, {sehir}'de yaşıyor.")
3
4   # Sıralı argümanlar
5   bilgi("Ahmet", 15, "İstanbul")
6
7   # Anahtar kelimeli argümanlar (sıra önemli değil)
8   bilgi(sehir="Ankara", ad="Ayşe", yas=16)
9
10  # Karışık (sıralı önce, sonra anahtarlı)
11  bilgi("Zeynep", sehir="İzmir", yas=14)
```

EKRAN ÇIKTISI
Ahmet, 15 yaşında, İstanbul'de yaşıyor.
Ayşe, 16 yaşında, Ankara'de yaşıyor.
Zeynep, 14 yaşında, İzmir'de yaşıyor.

Anahtar kelimeli argümanlarda hangi değerin hangi parametreye gideceği açıkça belirtildiği için argümanların sırası önemli değildir. Bu özellik çok parametreli fonksiyonlarda kodu daha anlaşılır kılar.

SIRA SİZDE

İki sayı ve bir işlem adının verildiği hesap_yap(sayi1, sayi2, islem) adında bir fonksiyon yazınız. Fonksiyonu farklı argümanlarla en az üç kez çağırarak test ediniz.

1. islem "toplama" ise iki sayının toplamını yazdırsın.

2. islem "carpma" ise iki sayının çarpımını yazdırsın.

Değer Döndüren Fonksiyonlar

Bir fonksiyon hesapladığı sonucu print() komutuyla ekrana yazdırır. Ancak yazdırmak, o veriyi kaydetmek anlamına gelmez. Eğer hesaplanan sonuç daha sonra başka matematiksel işlemlerde veya koşullarda tekrar kullanılacaksa veriyi ekrana yazdırmak yerine **return (döndür)** anahtar kelimesiyle programın kendisine geri gönderilmesi gerekir. return çalıştığında fonksiyon **sona erer.** İfadeden sonraki satırlar çalışmaz.

13. ÖRNEK UYGULAMA

Amaç: return ile değer döndüren bir fonksiyon yazmak

```
1  def kare_al(sayi):
2      sonuc = sayi * sayi
3      return sonuc
4
5  # Döndürülen değer bir değişkene atanır
6  deger = kare_al(5)
7  print("5'in karesi:", deger)
8
9  # Döndürülen değer doğrudan kullanılabilir
10 print("3'ün karesi:", kare_al(3))
```

EKRAN ÇIKTISI
5'in karesi: 25
3'ün karesi: 9

İşlemin temel farkı şöyledir: print() sonucu ekrana yazdırılır, return ifadesiyle sonuç fonksiyonu çağıran programa geri gönderilir. Döndürülen değer bir değişkene atanabilir veya başka bir işlemde kullanılabilir.

14. ÖRNEK UYGULAMA

Amaç: Değer döndüren fonksiyonun sonucunu başka işlemde kullanmak

```
def topla(a, b):
    return a + b      # Sonucu döndürür

# Çağrı ile gelen sonuç bir değişkene atanabilir
sonuc = topla(5, 3)
print(sonuc)          # 8

# Veya başka bir hesapta kullanılabilir
toplam = topla(2, 3) + topla(4, 5)
print(toplam)         # 14

# Veya doğrudan yazdırılabilir
print(topla(10, 20))  # 30
```

EKRAN ÇIKTISI
8
14
30

15. ÖRNEK UYGULAMA

Amaç: Birden fazla değer döndüren fonksiyon kullanmak

Bir fonksiyon çağrıldığında virgülle ayırılarak birden fazla değer döndürülebilir.

```
def istatistik(notlar):
    #Bir not listesinin min, max ve ortalama değerini döndürür.
    en_dusuk = min(notlar)
    en_yuksek = max(notlar)
    ortalama = sum(notlar) / len(notlar)
    return en_dusuk, en_yuksek, ortalama

# Demet açma ile birden fazla değişkene atama
dusuk, yuksek, ort = istatistik([75, 82, 68, 90, 55])
print(f"Min: {dusuk}, Max: {yuksek}, Ortalama: {ort}")
```

EKRAN ÇIKTISI
Min: 55, Max: 90, Ortalama: 74.0

16. ÖRNEK UYGULAMA

Amaç: Koşula göre farklı değer döndüren fonksiyon kullanmak

Bir fonksiyon, içindeki koşula göre farklı değerler döndürülebilir. Aşağıdaki fonksiyonla bir notun harf karşılığına çevrilmesi gerçekleşir.

```
1  def harf_notu(puan):
2      if puan >= 85:
3          return "AA"
4      elif puan >= 70:
5          return "BA"
6      elif puan >= 50:
7          return "CC"
8      else:
9          return "FF"
10
11 print("90 puan:", harf_notu(90))
12 print("75 puan:", harf_notu(75))
13 print("40 puan:", harf_notu(40))
```

EKRAN ÇIKTISI

Min: 55, Max: 90, Ortalama: 74.0

Fonksiyonla **puan** değerine göre dört farklı return ifadesinden biri çalıştırılır. return çalıştığı anda fonksiyon sonlanır ve bu yüzden uygun olan ilk return değeri döndürülür.

Yeni başlayan programcıların en sık karıştırdığı konulardan biri **print()** ile **return** arasındaki farktır. Tablo 5.2'de bu iki kavram karşılaştırılmıştır.

Tablo 5.2: print() Fonksiyonu İle return Arasındaki Fark

Özellik	print()	return
Görevi	Değeri ekrana yazar.	Değeri programa geri gönderir.
Sonuç kullanılabilir mi?	Hayır, sadece görüntülenir.	Evet, başka işlemde kullanılır.
Fonksiyonu sonlandırır mı?	Hayır	Evet
Değişkene atanabilir mi?	Hayır (None döner)	Evet

BİLGİ KUTUSU

Python'da aslında her fonksiyon bir değer döndürür. Eğer fonksiyonda return anahtar kelimesi kullanarak özel bir değer döndürülmezse, Python arka planda otomatik olarak **None** değerini döndürür. Bir print() işleminin sonucu değişkene atandığında değerin None çıkmasının asıl sebebi budur.

17. ÖRNEK UYGULAMA

Amaç: print() ve return farkını gözlemlemek

```
1  def topla_yazdir(a, b):
2      print(a + b)        # Sadece ekrana yazar
3
4  def topla_dondur(a, b):
5      return a + b        # Değeri geri verir
6
7  sonuc1 = topla_yazdir(3, 4)    # Ekrana 7 yazar
8  sonuc2 = topla_dondur(3, 4)    # 7 değerini döndürür
9
10 print("sonuc1 değişkeni:", sonuc1)
11 print("sonuc2 değişkeni:", sonuc2)
```

EKRAN ÇIKTISI

```
Min: 55, Max: 90, Ortalama: 74.0
```

topla_yazdir fonksiyon sonucu ekrana yazdırılır ama geri döndürülmediği için sonuc1 değişkenine **None** atanır. **topla_dondur** ile sonuç döndürüldüğü için sonuc2 değişkenine gerçek değer (7) atanır.

SIRA SİZDE

Bir dairenin yarıçapını parametre olarak alan ve dairenin alanını döndüren daire_alani (yaricap) adında bir fonksiyon yazınız. (Alan = 3.14 * yarıçap * yarıçap)

Fonksiyonu kullanarak yarıçapı 5 ve 10 olan iki dairenin alanlarını hesaplayınız. Ardından bu iki alanın toplamını ekrana yazdırınız. Fonksiyonun sonucu döndürdüğüne dikkat ediniz.

18. ÖRNEK UYGULAMA

Amaç: Değer döndüren fonksiyonlarla bir hesaplama zinciri kurmak

Değer döndüren fonksiyonlarda bir fonksiyonun çıktısı diğerinin girdisi olabilir. Aşağıdaki örnekte bir ürünün net fiyatı adım adım hesaplanmıştır.

```
1  def indirim_uygula(fiyat, oran):
2      return fiyat - (fiyat * oran / 100)
3
4  def vergi_ekle(fiyat):
5      return fiyat * 1.20
6
7  # İlk fonksiyonun çıktısı, ikinciye girdi oluyor
8  indirimli = indirim_uygula(1000, 25)
9  net_fiyat = vergi_ekle(indirimli)
10
11 print("İndirimli fiyat:", indirimli)
12 print("Vergili net fiyat:", net_fiyat)
```

EKRAN ÇIKTISI
İndirimli fiyat: 750.0
Vergili net fiyat: 900.0

Örnekte **indirim_uygula** fonksiyonunun döndürdüğü değer (750.0), **vergi_ekle** fonksiyonuna girdi olarak verilir. Değer döndüren fonksiyonlar zincirleme kullanılarak karmaşık hesaplamalar adım adım yapılır.

Değişken Kapsamı

Bir fonksiyon içinde tanımlanan değişkenler yalnızca o fonksiyonun içinde geçerlidir. Bu duruma **değişken kapsamı** (scope) denir. Kapsam, bir değişkenin program içinde nerelerden erişilebileceğini belirler.

Yerel Değişken: Bir fonksiyonun içinde tanımlanan değişkendir. Yalnızca tanımlandığı fonksiyon içinde kullanılabilir. Değişkene fonksiyon dışından erişilemez.

Global Değişken: Fonksiyonun dışında tanımlanan değişkendir. Değişkene program genelinde ve tüm fonksiyonların içinde erişilebilir.

Yerel ve global değişkenler arasındaki ilişki Görsel 5.3'te gösterilmiştir.



Görsel 5.3: Yerel ve global değişkenler

BİLGİ KUTUSU

Programda yerel bir değişkene dışarıdan ulaşmaya çalışıldığında alınan **NameError** hatası, Python'un "Benim genel (global) kayıt defterimde böyle bir değişken adı yok." deme şeklidir. Ana programdan bir değişken çağırıldığında Python yalnızca global alandaki değişkenleri görebilir. Fonksiyonlar yerel değişkenlerin adı ve varlığını bilemez. Bu nedenle fonksiyon içinde oluşturulan bir değere dışarıdan doğrudan ulaşmaya çalışmak her zaman hataya yol açar.

19. ÖRNEK UYGULAMA

Amaç: Yerel değişkenin kapsamını gözlemlemek

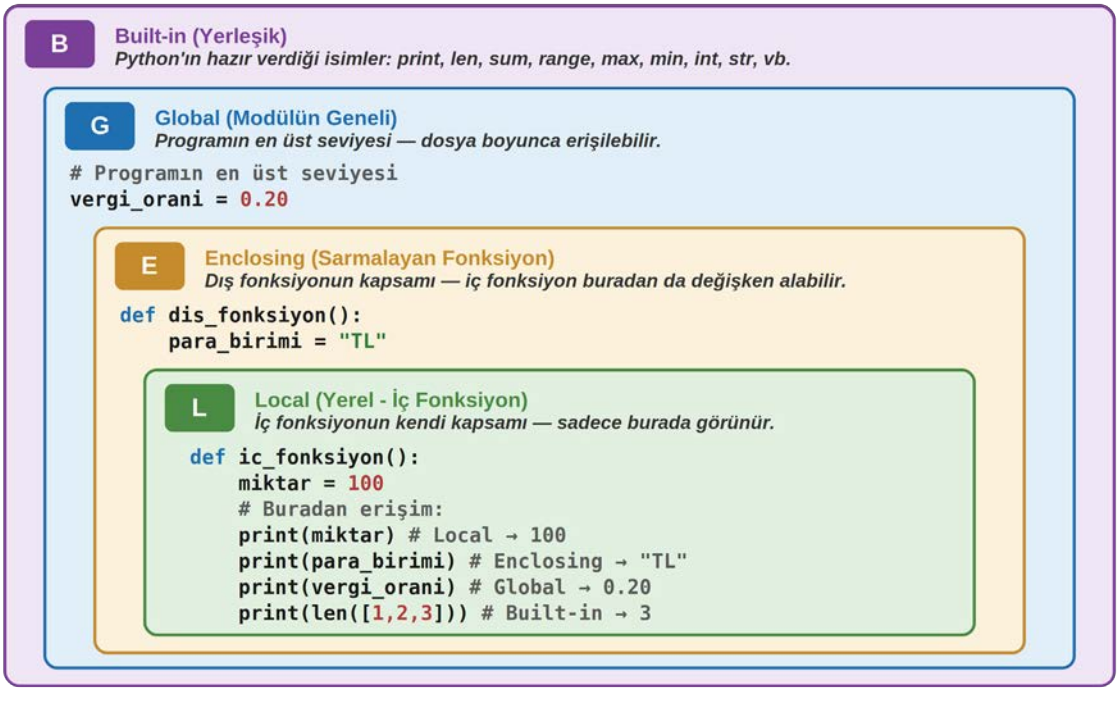
```
1   mesaj = "Dışarıdan merhaba"      # GLOBAL değişken
2
3   def fonksiyonum():
4       yerel = "İçeriden merhaba"   # YEREL değişken
5       print(mesaj)                 # Globalı okur - OK
6       print(yerel)                 # Yereli okur - OK
7
8   fonksiyonum()
9   print(mesaj)                     # Globalı okur - OK
10  # print(yerel)                   # HATA: yerel global değil!
```

EKRAN ÇIKTISI
Dışarıdan merhaba
İçeriden merhaba
Dışarıdan merhaba

BİLGİ KUTUSU

Python bir değişkene erişirken **LEGB** sırası izlenir. Bu kısaltmanın anlamı şöyledir:

- **L (Local)**-Yerel kapsam: Mevcut fonksiyon
- **E (Enclosing)**-Çevreleyen kapsam: İç içe fonksiyonlarda dış fonksiyon
- **G (Global)**-Genel kapsam: Modül seviyesi
- **B (Built-in)**-Yerleşik: Python'un kendi tanımladığı (print, len gibi



Bir fonksiyon içinde yer alan global değişkenin değeri değiştirmek istenirse **global** anahtar sözcüğünün kullanılması gerekir.

20. ÖRNEK UYGULAMA

Amaç: Fonksiyon içinde global değişken tanımlamak

```
1   sayac = 0    # GLOBAL
2
3   def artir():
4       global sayac          # "Bu sayac global değişken!"
5       sayac = sayac + 1
6
7   artir()
8   artir()
9   artir()
10  print(sayac)              # 3
11
12  # global olmadan
13  sayi = 5
14  def degistir():
15      sayi = 10             # Bu YENİ bir yerel değişken!
16      print("Fonksiyon içinde:", sayi)
17
18  degistir()                # 10
19  print("Fonksiyon dışında:", sayi)    # 5 - global değişmedi!
```

EKRAN ÇIKTISI
3
Fonksiyon içinde: 10
Fonksiyon dışında: 5

BİLGİ KUTUSU

Mümkün olduğunca global değişken kullanmaktan kaçınınız. Global değişkenler programın farklı yerlerinde değişebileceğinden hatayı bulmak zorlaşır. Bunun yerine fonksiyonlara değerleri **parametre** olarak gönderiniz ve sonucu **return** ile geri alınız.

21. ÖRNEK UYGULAMA

Amaç: Fonksiyon her çağrıldığında yerel değişkenleri yeniden oluşturmak

Bir fonksiyonun yerel değişkenleri, fonksiyon her çağrıldığında yeniden oluşturulur. Önceki çağrıdan hiçbir iz bırakılmaz.

```
1  def sayac_dene():
2      sayi = 0          # her çağrıda yeniden 0 olur
3      sayi = sayi + 1
4      print("Bu çağrıdaki değer:", sayi)
5
6  sayac_dene()
7  sayac_dene()
8  sayac_dene()
```

EKRAN ÇIKTISI
Bu çağrıdaki değer: 1
Bu çağrıdaki değer: 1
Bu çağrıdaki değer: 1

BİLGİ KUTUSU

Değişken kapsamı, programlamada hata olasılığını azaltan önemli bir kavramdır. Her fonksiyonun kendi yerel değişkenleri olduğu için bir fonksiyondaki değişken adı başka bir fonksiyondaki aynı isimli değişkeni etkilemez. Bu, büyük programlarda fonksiyonların birbirinden bağımsız çalışmasını sağlar.

22. ÖRNEK UYGULAMA

Amaç: Farklı fonksiyonlarda aynı isimli yerel değişkenler kullanmak

İki ayrı fonksiyon tarafından aynı isimli yerel değişkenler kullanılabilir, bu değişkenler birbirinden tamamen bağımsızdır.

```
1  def fonksiyon_a():
2      sonuc = 10
3      print("A fonksiyonunda sonuc:", sonuc)
4
5  def fonksiyon_b():
6      sonuc = 99
7      print("B fonksiyonunda sonuc:", sonuc)
8
9  fonksiyon_a()
10 fonksiyon_b()
```

EKRAN ÇIKTISI
A fonksiyonunda sonuc: 10
B fonksiyonunda sonuc: 99

SIRA SİZDE

Aşağıdaki kodları inceleyiniz ve çalıştırıldığında üreteceği çıktıyı tahmin ediniz.

1.
```
def fonksiyon_a():
mesaj = “Merhaba”

print(mesaj)
```

2.
```
def fonksiyon_b():
 print(mesaj)


fonksiyon_a()

fonksiyon_b()
```

3. Aşağıdaki soruları cevaplayınız:

a) fonksiyon_a() çağrıldığında ne yazılır?

b) fonksiyon_b() çağrıldığında ne olur ve neden?

c) mesaj değişkeni hangi fonksiyonun yerel değişkenidir?

Lambda Fonksiyonları

Tek bir işlem yapan kısa ve geçici bir fonksiyona ihtiyaç duyulduğunda Python'da **lambda fonksiyonları** kullanılır. Lambda fonksiyonu tek satırda yazılabilen, **isimsiz** (anonim) bir fonksiyondur.

Lambda fonksiyonunun genel söz dizimi şu şekildedir:

```
degisken = lambda parametreler: dondurulecek_ifade
```

Normal bir fonksiyon ile lambda fonksiyonu karşılaştırması Tablo 5.2'de verilmiştir.

Tablo 5.2: print() Fonksiyonu İle return Arasındaki Fark

Özellik	Normal Fonksiyon	Lambda Fonksiyonu
Tanımlama	def anahtar sözcüğü ile	lambda anahtar sözcüğü ile
İsim	Mutlaka bir adı vardır.	İsimsizdir (anonim).
Uzunluk	Birden çok satır olabilir.	Tek satırdır.
return kullanımı	Açıkça yazılır.	Otomatiktir, yazılmaz.
Kullanım	Karmaşık işler için	Kısa, basit işler için

23. ÖRNEK UYGULAMA

Amaç: Bir lambda fonksiyonu tanımlamak ve kullanmak

```
1   topla = lambda a, b: a + b
2   tek_mi = lambda x: x % 2 == 1
3   selam = lambda ad: f"Merhaba, {ad}!"
4
5   print(topla(3, 5))        # 8
6   print(tek_mi(7))          # True
7   print(selam("Ahmet"))     # Merhaba, Ahmet!
```

EKRAN ÇIKTISI

```
8
True
Merhaba, Ahmet!
```

24. ÖRNEK UYGULAMA

Amaç: Bir lambda fonksiyonu tanımlamak ve kullanmak

```
1   topla = lambda a, b: a + b
2   buyuk_olan = lambda a, b: a if a > b else b
3
4   print("Toplam:", topla(12, 8))
5   print("Büyük olan:", buyuk_olan(15, 23))
```

EKRAN ÇIKTISI

```
Toplam: 20
Büyük olan: 23
```

BİLGİ KUTUSU

Lambda fonksiyonları yalnızca tek bir ifade içerebilir. Birden fazla satır kod veya karmaşık işlemler için kullanılamaz. Karmaşık işler için def ile normal fonksiyon tanımlanmalıdır.

25. ÖRNEK UYGULAMA

Amaç: Lambda fonksiyonunu bir liste işleminde kullanmak

Lambda fonksiyonlarının özellikle bir liste üzerinde hızlı işlemler yapmak için elverişli olduğu belirtilir. Aşağıdaki örnekte bir sayının vergili fiyatını hesaplayan lambda fonksiyonunun, farklı fiyatlara uygulandığı görülür.

```
1  vergi_ekle = lambda fiyat: fiyat * 1.20
2
3  fiyatlar = [100, 250, 500]
4  for f in fiyatlar:
5      print(f, "TL ürünün vergili fiyatı:", vergi_ekle(f))
```

EKRAN ÇIKTISI
100 TL ürünün vergili fiyatı: 120.0
250 TL ürünün vergili fiyatı: 300.0
500 TL ürünün vergili fiyatı: 600.0

BİLGİ KUTUSU

filter() yerleşik fonksiyonu, bir veri koleksiyonundaki elemanları belirli bir koşula göre süzmek için kullanılır. Koşulu belirleyen bir fonksiyona ve süzülecek veri listesine ihtiyaç duyar.

filter() fonksiyonu kullanılırken koşulu def ile tanımlamak yerine genellikle tek satırlık pratik lambda fonksiyonları kullanılabilir. İşlem sonucunda sadece koşulu sağlayan elemanlar ayrılır ve list() komutuyla yeni bir listeye dönüştürülür.

26. ÖRNEK UYGULAMA

Amaç: Lambda fonksiyonu kullanarak filtreleme yapmak

```
1  sayilar=[1,2,3,4,5,6,7,8,9,10]
2  filtrelenmis_sayilar=list(filter(lambda x: x%2==0, sayilar))
3  print("sayilar:", sayilar)
4  print("filtrelenmis sayilar:", filtrelenmis_sayilar)
```

EKRAN ÇIKTISI
sayilar: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
filtrelenmis sayilar: [2, 4, 6, 8, 10]

Bu lambda fonksiyonu ile tek satırda bir koşul kontrolü yapılır ve sonuca göre farklı bir metin döndürülür. Lambda fonksiyonları tek satır sınırı, bu tür kısa koşullu ifadelerle esnek biçimde kullanılır.

27. ÖRNEK UYGULAMA

Amaç: Normal fonksiyon ve lambda fonksiyonunu birlikte değerlendirmek

Aşağıdaki örnekle aynı işin hem normal fonksiyonla hem de lambda ile nasıl yapıldığı yan yana gösterilir. İşin, bir öğrencinin geçip geçmediğinin belirlenmesi olduğu belirtilir.

```
1  # Normal fonksiyon ile
2  def gecti_mi_normal(puan):
3      if puan >= 50:
4          return "Geçti"
5      else:
6          return "Kaldı"
7
8  # Lambda ile (tek satır)
9  gecti_mi_lambda = lambda puan: "Geçti" if puan >= 50 else "Kaldı"
10
11 print("Normal fonksiyon:", gecti_mi_normal(65))
12 print("Lambda fonksiyon:", gecti_mi_lambda(65))
```

EKRAN ÇIKTISI
Normal fonksiyon: Geçti
Lambda fonksiyon: Geçti

SIRA SİZDE

Aşağıdaki işlemleri yapan lambda fonksiyonu yazarak ana programdan çağırınız. Yazdığınız lambda fonksiyonlarını farklı değerlerle çağırarak test ediniz.

1. Bir sayının iki katını döndüren iki_kat fonksiyonu:

2. Bir sayının çift olup olmadığını (True/False) döndüren cift_mi fonksiyonu:

3. Verilen iki sayıdan küçük olanı döndüren kucuk_olan fonksiyonu:

1. UYGULAMA

Personelin Maaşını Hesaplayan Fonksiyon Geliştiriyorum

Süre: 40 dakika

Amaç: Bir şirketteki personelin maaşını hesaplayan fonksiyon geliştirmek

Yönerge: Aşağıdaki adımları takip ederek uygulamayı geliştiriniz.

1. Adım: Editörünüzde **maas_hesaplama.py** adında yeni bir dosya oluşturunuz. Problemi parçalarına ayırınız. Maaş hesabı hangi bilgilere ihtiyaç duyar?

2. Adım: Normal mesai ücretini hesaplayan **normal_ucret** adında bir fonksiyon yazınız. Bu fonksiyon; saatlik ücreti ve çalışılan saati parametre olarak alsın, normal ücreti return ile döndürsün.

3. Adım: Fazla mesai ücretini hesaplayan **mesai_ucreti** adında ikinci bir fonksiyon yazınız. Fazla mesai saatleri, normal saatlik ücretin 1,5 katı üzerinden ödenir.

4. Adım: Bu iki fonksiyonu birlikte kullanan **toplam_maas** adında üçüncü bir fonksiyon yazınız. Bu fonksiyon, normal ücret ile mesai ücretini toplayarak toplam maaşı döndürsün.

5. Adım: Programınızı, kullanıcıdan bir personelin saatlik ücretini, normal çalışma saatini ve fazla mesai saatini **input()** ile alacak şekilde düzenleyiniz. Alınan verileri toplam_maas fonksiyonuna göndererek sonucu ekrana yazdırınız.

6. Adım: Programınızı en az üç farklı personel için şu şekilde test ediniz:

a) fazla mesai yapmayan

b) az miktarda mesai yapan

c) çok mesai yapan bir personel

Sonuçların doğruluğunu elle hesaplayarak kontrol ediniz.

Test Edilecek Durum	Saatlik Ücret	Normal Çalışma Saati	Fazla Mesai Saati
Fazla Mesai Yapmayan Personel	150	40	0
Fazla Mesai Yapan Personel	150	40	4
Çok Fazla Mesai Yapan Personel	150	40	15

7. Adım: Tamamlanan programınızı README dosyasıyla birlikte dijital deponuza yükleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

28. ÖRNEK UYGULAMA

Amaç: Birden fazla fonksiyonu bir arada kullanan program yazmak

```
1  def ortalama_bul(n1, n2, n3):
2      return (n1 + n2 + n3) / 3
3
4  def durum_belirle(ortalama):
5      if ortalama >= 50:
6          return "Geçti"
7      else:
8          return "Kaldı"
9
10 def rapor_yaz(ad, ortalama, durum):
11     print("Öğrenci:", ad)
12     print("Ortalama:", ortalama)
13     print("Durum:", durum)
14
15 # Fonksiyonlar birlikte kullanılıyor
16 ort = ortalama_bul(60, 70, 80)
17 snc = durum_belirle(ort)
18 rapor_yaz("Elif", ort, snc)
```

EKRAN ÇIKTISI
Öğrenci: Elif
Ortalama: 70.0
Durum: Geçti

Özyinelemeli Fonksiyonlar

Fonksiyonun çözüm sürecinde kendi kendini çağırması yöntemine **özyineleme (recursion)** denir. Bir problemi çözmek için o problemin daha küçük bir versiyonunu çözme mantığına dayanır.

Büyükten küçüğe iç içe konulmuş kutular var. En büyük kutu açıldığında içinden daha küçük bir kutu çıkar, bu kutu açıldığında ise daha da küçük bir kutuya ulaşılır. Bu işlem **en küçük kutuya** ulaşılana kadar tekrarlanır. Her adımda gerçekleştirilen işlem aynıdır. Yalnızca kutuların boyutu değişir. Özyineleme de buna benzer şekilde çalışır.

Özyineleme ve döngü, aslında aynı amaca hizmet eden iki farklı yöntemdir. İkisi de bir işi **tekrar tekrar** yapar. Döngüde tekrar, for veya while yapısıyla sağlanır. Özyinelemede ise tekrar, fonksiyonun kendini yeniden çağırmasıyla gerçekleşir.

Döngüde tekrarın ne zaman duracağını bir **koşul** belirler. Özyinelemede ise tekrarın ne zaman duracağını **taban durum** belirler. Döngünün koşulu hiç sağlanmazsa sonsuz döngü oluşur. Özyinelemenin taban durumuna hiç ulaşılmazsa sonsuz özyineleme oluşur. Her iki durumda da program düzgün çalışmaz.

29. ÖRNEK UYGULAMA

Amaç: Bir fonksiyonun kendini çağırma sürecini gözlemlemek

```
1  def merhaba_de(sayi):
2      if sayi == 0:
3          print("Özyineleme bitti.")
4          return
5      print("Merhaba! Kalan tekrar:", sayi)
6      merhaba_de(sayi - 1)
7
8  merhaba_de(3)
```

EKRAN ÇIKTISI
Merhaba! Kalan tekrar: 3
Merhaba! Kalan tekrar: 2
Merhaba! Kalan tekrar: 1
Özyineleme bitti.

Fonksiyon her çağrıldığında bir mesajın yazıldığı ve **sayi** değerinin bir azaltılarak fonksiyonun yeniden çağrıldığı görülür. **sayi** değeri 0 olduğunda taban duruma ulaşılır ve özyineleme tamamlanır. Özyineleme küçülerek tekrarlanan bir çağrı zinciridir.

Özyinelemenin Temel Ögeleri

Her özyinelemeli fonksiyonun **iki temel ögesi** olmalıdır. Bu ögelerden biri eksik olursa fonksiyon doğru çalışmaz.

Taban Durum (Base Case): Özyinelemenin **duracağı** en basit durumdur. Fonksiyon bu duruma ulaştığında kendini artık çağırmaz, doğrudan bir sonuç döndürür. Taban durum, özyinelemenin "dur" işaretidir.
Özyinelemeli Adım (Recursive Step): Fonksiyonun kendini, problemin **daha küçük bir parçasıyl**a çağırdığı kısımdır. Her çağrı problemi biraz daha küçültür ve sonunda taban duruma ulaşılır.

30. ÖRNEK UYGULAMA

Amaç: Basit bir özyinelemeli fonksiyon yazmak

```
1  def geri_say(sayi):
2      if sayi < 0:          # Taban durum: dur
3          print("Bitti!")
4          return
5      print(sayi)
6      geri_say(sayi - 1)    # Özyinelemeli adım: kendini daha küçükle çağır
7
8  geri_say(5)
```

EKRAN ÇIKTISI

```
5
4
3
2
1
0
Bitti!
```

31. ÖRNEK UYGULAMA

Amaç: Özyinelemeli fonksiyonun çağrı zincirini izlemek

Bir özyinelemeli fonksiyonun nasıl çalıştığının anlaşılması için her çağrıda bir mesaj yazdırılır ve çağrı zinciri izlenir.

```
1  def toplam(n):
2      print("toplam(" + str(n) + ") çağrıldı")
3      if n == 1:                  # Taban durum
4          print("Taban duruma ulaşıldı")
5          return 1
6      sonuc = n + toplam(n - 1)  # Özyinelemeli adım
7      print("toplam(" + str(n) + ") sonucu:", sonuc)
8      return sonuc
9
10 print("SONUÇ:", toplam(4))
```

EKRAN ÇIKTISI

```
toplam(4) çağrıldı
toplam(3) çağrıldı
toplam(2) çağrıldı
toplam(1) çağrıldı
Taban duruma ulaşıldı
toplam(2) sonucu: 3
toplam(3) sonucu: 6
toplam(4) sonucu: 10
SONUÇ: 10
```

32. ÖRNEK UYGULAMA

Amaç: Taban durumu olmayan bir fonksiyonun tehlikesini göstermek

Bir özyinelemeli fonksiyonda taban durumun unutulması hâlinde ne olacağının gösterildiği belirtilir. Aşağıdaki fonksiyonda taban durumun bulunmadığı ifade edilir.

```
1  def sonsuz_say(n):
2      print(n)
3      sonsuz_say(n + 1)   # Taban durum yok!
4
5  #sonsuz_say(1)   # Bu satır çalıştırılırsa program çöker
```

EKRAN ÇIKTISI

```
1
2
3
...
RecursionError: maximum recursion depth exceeded
```

Bu fonksiyonda **taban durum** bulunmadığında fonksiyon giderek artan değerlerle çağrılır. Belirli bir çağrı derinliğine ulaşıldığında **RecursionError** hatası verilir ve programın çöktüğü belirtilir.

SIRA SİZDE

Aşağıdaki özyinelemeli fonksiyonları inceleyiniz. Her biri için taban durumu ve özyinelemeli adımı belirleyiniz. Hangisinde taban durum eksik veya hatalıdır?

1.
```
def f(n):
    if n == 0:
        return 0
    return n + f(n - 1)
```

2.
```
def g(n):
    return n * g(n - 1)
```

3. 2. sorudaki fonksiyon çalıştırılırsa ne olur? Eksik olan ögeyi ekleyerek fonksiyonu düzeltiniz.

Özyinelemeli Fonksiyon Yazımı

Özyinelemenin en bilinen örneği **faktöriyel** hesabıdır. Bir sayının faktöriyeli, o sayıdan 1'e kadar olan tüm sayıların çarpımıdır.

Faktöriyel özyinelemeli olarak şöyle tanımlanır: Bir sayının faktöriyeli, o sayı ile bir küçüğünün faktöriyelinin çarpımıdır. Kısaca **n! = n × (n-1)!** şeklinde ifade edilir. Örneğin 4! = 4 × 3 × 2 × 1 = 24 olarak bulunur.

6!=6 × 5× 4 × 3 × 2 × 1 = 120 olarak bulunur.

33. ÖRNEK UYGULAMA

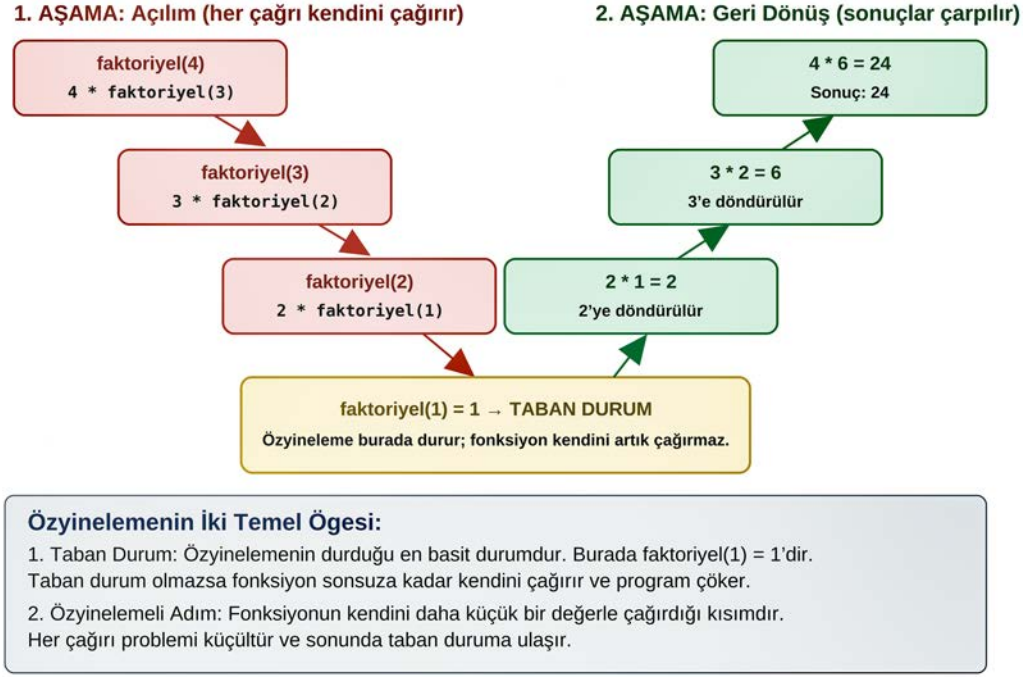
Amaç: Faktöriyel hesabını özyinelemeli fonksiyonla yapmak

```
1   # Matematiksel tanım:
2   # n! = n × (n-1)!          (Özyinelemeli durum)
3   # 0! = 1                   (Taban durumu)
4
5   def faktoriyel(n):
6       if n == 0 or n == 1:        # TABAN DURUMU
7           return 1
8       return n * faktoriyel(n - 1)  # ÖZYİNELEMELİ DURUM
9
10  print(faktoriyel(5))
11
12  # Adım adım nasıl çalışır:
13  # faktoriyel(5) = 5 * faktoriyel(4)
14  #               = 5 * 4 * faktoriyel(3)
15  #               = 5 * 4 * 3 * faktoriyel(2)
16  #               = 5 * 4 * 3 * 2 * faktoriyel(1)
17  #               = 5 * 4 * 3 * 2 * 1
18  #               = 120
```

EKRAN ÇIKTISI

```
120
```

faktöriyel fonksiyonu negatif sayıyla çağırılırsa özyineleme bin defa tekrarlanarak durur. **faktöriyel(4)** çağrıldığında fonksiyonun sırasıyla faktoriyel(3), faktoriyel(2) ve faktoriyel(1) ile çağrıldığı belirtilir. faktoriyel(1) taban duruma ulaştığında 1'in döndürüldüğü ve sonuçların geriye doğru çarpılarak sonucun (24) olduğu görülür. Bu sürecin tamamı Görsel 5.4'te adım adım gösterilmiştir.



Görsel 5.4: Faktöriyel(4) hesabının adımları

SIRA SİZDE

Bir sayının rakamlarının toplamını özyineleme ile hesaplayan rakam_topla(sayi) fonksiyonunu yazınız. Örneğin 425 sayısı için sonuç 4+2+5 = 11 olmalıdır. Fonksiyonu 425 ve 1234 sayılarıyla test ediniz.

BİLGİ KUTUSU

Özyinelemeli fonksiyon yazarken şu üç soru sorulur:

1. Problemin en basit hâli nedir? Bu, taban durumdur.
2. Problemi nasıl daha küçük bir parçaya bölebilirim? Bu, özyinelemeli adımdır.
3. Küçük parçanın sonucunu kullanarak büyük problemi nasıl çözerim?

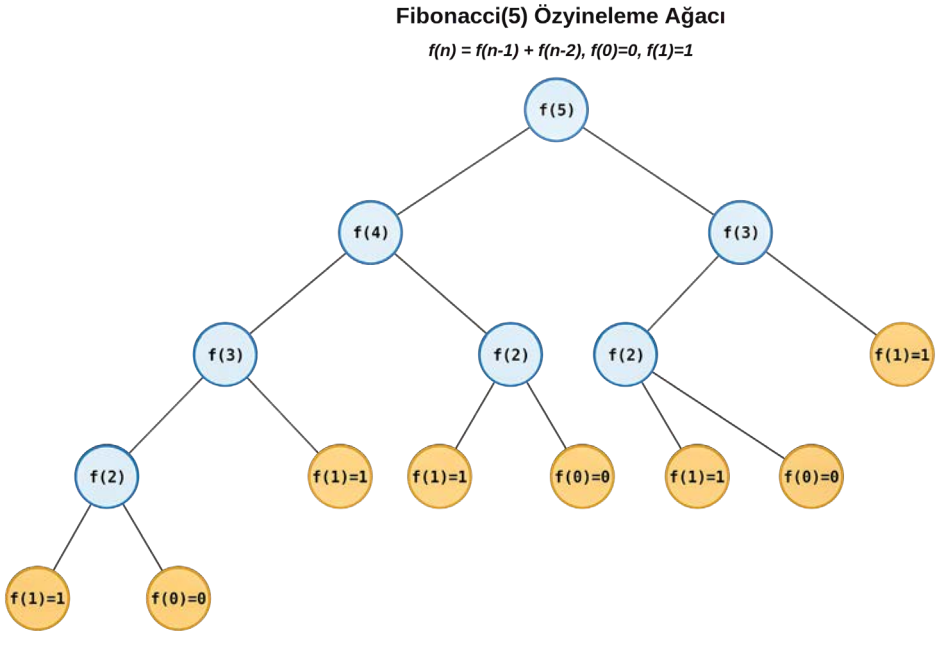
Üç soru, özyinelemeli düşünmenin anahtarıdır.

34. ÖRNEK UYGULAMA

Amaç: Fibonacci dizisini özyineleme ile hesaplamak

Fibonacci dizisinin, her sayının kendinden önceki iki sayının toplamı olduğu ünlü bir sayı dizisi: 0, 1, 1, 2, 3, 5, 8, 13...

Fibonacci dizisinin matematiksel tanımı Görsel 5.6'dan inceleyerek program yazılır.



Görsel 5.6: Fibonacci dizisinin özyinelemeli yapısı

```
# Matematiksel tanım:
# F(0) = 0
# F(1) = 1
# F(n) = F(n-1) + F(n-2)    (n >= 2)

def fibonacci(n):
    if n == 0:        # TABAN 1
        return 0
    if n == 1:        # TABAN 2
        return 1
    return fibonacci(n - 1) + fibonacci(n - 2)

# Dizinin ilk 10 elemanı
for i in range(10):
    print(fibonacci(i), end=" ")
```

EKRAN ÇIKTISI

0 1 1 2 3 5 8 13 21 34

fibonacci(0)=0 ve fibonacci(1)=1 örnekte olduğu gibi **iki taban durum** bulunur. Özyinelemeli adımda fonksiyon **iki kez** çağrılır. Bu örnekle bazı problemlerde birden fazla taban durum veya birden fazla özyinelemeli çağrı gerekebileceğinin gösterildiği belirtilir.

SIRA SİZDE

Aşağıdaki özyinelemeli fonksiyonu inceleyiniz ve çağrı(4) için adım adım nasıl çalışacağını bir kâğıda yazınız.

```
def cagri(n):
    if n == 0:
        return 0
    return n + cagri(n - 1)
```

Aşağıdaki soruları cevaplayınız:

1. cagri(4) hangi sonucu üretir?

2. Fonksiyon kendini kaç kez çağırır?

3. Taban durum hangi çağrıda gerçekleşir?

35. ÖRNEK UYGULAMA

Amaç: Bir listedeki elemanları özyineleme ile ekrana yazdırmak

Özyineleme, bir listenin elemanlarını tek tek işlemek için de kullanılabilir. Aşağıdaki örnekte fonksiyon listenin elemanları sırayla ekrana yazdırılır.

```
1  def liste_yaz(liste):
2      if len(liste) == 0:        # Taban durum: boş liste
3          return
4      print(liste[0])            # İlk elemanı yazdır
5      liste_yaz(liste[1:])       # Geri kalanı kendine gönder
6
7  meyveler = ["elma", "armut", "kiraz", "muz"]
8  liste_yaz(meyveler)
```

EKRAN ÇIKTISI
elma
armut
kiraz
muz

Fonksiyonla her adımda listenin **ilk elemanı** yazdırılır ve **geri kalan liste** kendine gönderilir. Liste boşaldığında taban duruma ulaşılır ve özyineleme sona erdirilir. Bu örnekle özyinelemenin liste gibi koleksiyon veri yapılarıyla nasıl kullanıldığı gösterilir.

Özyineleme güçlü bir tekniktir ancak her problem için en uygun çözüm olmayabilir. Bir programcının özyinelemenin avantajlarını ve dezavantajlarını bilmesi, doğru durumda doğru tekniği seçmesini sağlar.

Avantajları

- Bazı problemler (faktöriyel, ağaç yapıları, iç içe yapılar) özyineleme ile çok daha doğal ve anlaşılır biçimde çözülür.
- Kod genellikle daha kısa ve zarif olur.
- Problemin matematiksel tanımına yakın bir çözüm sunar.

Dezavantajları

- Her fonksiyon çağrısı, bellekte yer kapladığı için çok derin özyineleme bellek tüketebilir.
- Döngülere göre genellikle daha yavaş çalışır.
- Taban durum unutulursa sonsuz özyineleme oluşur ve program çöker.

36. ÖRNEK UYGULAMA

Amaç: Aynı problemi döngü ve özyineleme ile çözmek

Aşağıda bir sayının faktöriyeli, iki farklı yöntemle hesaplanmıştır.

```
1  # Döngü ile
2  def faktoriyel_dongu(n):
3      sonuc = 1
4      for i in range(1, n + 1):
5          sonuc = sonuc * i
6      return sonuc
7
8  # Özyineleme ile
9  def faktoriyel_ozyineleme(n):
10     if n == 1:
11         return 1
12     return n * faktoriyel_ozyineleme(n - 1)
13
14 print("Döngü ile 5! =", faktoriyel_dongu(5))
15 print("Özyineleme ile 5! =", faktoriyel_ozyineleme(5))
```

EKRAN ÇIKTISI

```
Döngü ile 5! = 120
Özyineleme ile 5! = 120
```

Her iki yöntemle de aynı sonuç elde edilir. Hangisinin seçileceği, problemin yapısına ve programcının tercihine bağlıdır.

Problem doğal olarak **daha küçük alt problemlere** bölünebiliyorsa (faktöriyel, iç içe yapılar, ağaç gezme gibi) özyinelemeyle daha etkili bir çözüm sunulur. Problem basit bir tekrar gerektiriyorsa (bir listeyi baştan sona dolaşmak gibi) ve performans önemliyse döngü tercih edilir.

Genel bir kural olarak özyineleme, kodun kısa ve anlaşılır olmasını sağlar. Döngülerle daha hızlı ve daha az bellek kullanan çözümler sunulur. Deneyimli bir programcı, problemin yapısına bakarak doğru yöntemi seçer.

BİLGİ KUTUSU

Python'da özyineleme derinliği varsayılan olarak sınırlıdır (genellikle yaklaşık 1000 çağrı). Çok derin özyineleme gerektiren problemlerde program **RecursionError** hatası verebilir. Bu tür durumlarda döngü kullanmak daha güvenlidir. Bu durum, özyinelemenin bir dezavantajıdır.

5.2. MODÜLER PROGRAMLAMA

Büyük bir yazılım projesini daha küçük, bağımsız ve yeniden kullanılabilir parçalara (modüllere) ayırma yaklaşımına **modüler programlama** denir. Python'da bir modül .py uzantılı herhangi bir dosyadır.

Modüler programlamanın temel avantajları şunlardır:

- **Okunabilirlik:** Kod daha anlaşılır ve düzenli olur.
- **Yeniden Kullanım:** Aynı kod farklı projelerde kullanılabilir.
- **Bakım Kolaylığı:** Hata tespiti ve düzeltme daha kolaydır.
- **Takım Çalışması:** Farklı ekipler farklı modüller üzerinde çalışabilir.
- **Test Edilebilirlik:** Her modül ayrı ayrı test edilebilir.

Hazır Modülleri Kullanma

Python, kurulumla birlikte gelen kapsamlı bir standart kütüphane sunar. Bu modüller import anahtar kelimesiyle programa dahil edilebilir. Yaygın kullanılan standart modüller ve örnek kullanımları Tablo 5.3'te verilmiştir.

Tablo 5.3: Yaygın kullanılan standart modüller

Modül Adı	Açıklama	Örnek Kullanım
math	Matematiksel fonksiyonlar	math.sqrt(25), math.pi
random	Rastgele sayı üretme	random.randint(1,10)
os	İşletim sistemi işlemleri	os.getcwd(), os.listdir()
sys	Python yorumlayıcı bilgileri	sys.version, sys.argv
datetime	Tarih ve saat işlemleri	datetime.date.today()
collections	Gelişmiş veri yapıları	Counter, deque, OrderedDict

37. ÖRNEK UYGULAMA

Amaç: import komutu ile modül yüklemek

```
1   import math
2   # Modül adıyla fonksiyona erişim
3   kare_kok = math.sqrt(144)
4   pi_sayisi = math.pi
5
6   print(f'Karekök: {kare_kok}')
7   print(f'Pi sayısı: {pi_sayisi:.4f}')
```

EKRAN ÇIKTISI

```
Karekök: 12.0
Pi sayısı: 3.1416
```

Uzun modül adlarını kısaltmak veya çakışmaları önlemek için **takma ad (alias)** kullanılır.

38. ÖRNEK UYGULAMA

Amaç: import ... as ile takma ad kullanmak

```
1   import datetime as dt
2   import random as rnd
3
4   bugun = dt.date.today()
5   sayi  = rnd.randint(1, 100)
6
7   print(f'Bugünün tarihi: {bugun}')
8   print(f'Rastgele sayı: {sayi}')
```

EKRAN ÇIKTISI

```
Bugünün tarihi: 2026-06-15
Rastgele sayı: 24
```

Büyük modüllerin tamamını yüklemek yerine, yalnızca ihtiyaç duyulan bileşenler **from ... import** söz dizimiyle seçilerek içe aktarılabilir.

39. ÖRNEK UYGULAMA

Amaç: from .. import ile kütüphane kullanmak

```
1   from math import sqrt, pi
2   print("Doğrudan Pi kullanımı:", pi)
3   print("Doğrudan Karekök kullanımı:", sqrt(25))
```

EKRAN ÇIKTISI

```
Doğrudan Pi kullanımı: 3.141592653589793
Doğrudan Karekök kullanımı: 5.0
```

SIRA SİZDE

Bir liste tanımlayınız. random kütüphanesini kullanarak listeye 0-100 arası on adet sayı ekleyiniz.

1. enbuyukbul isimli bir fonksiyonla listedeki en büyük sayıyı buldurunuz.
2. ortalamabul isimli bir fonksiyonla listedeki sayıların ortalamasını buldurunuz.
3. Ana programdan iki fonksiyonu çağırarak sonuçları yazdırınız.

Modül Oluşturma

Programcı kendi modüllerini yazarak programlarında kullanabilir. Bunun için fonksiyonları içeren bir Python dosyası oluşturularak ana programla aynı klasöre kaydedilir.

40. ÖRNEK UYGULAMA

Amaç: Modül yazmak ve ana programdan çağırmak

1. **geometri.py** adında bir dosya oluşturulur.

```
1   import math
2   PI = math.pi
3
4   def daire_alani(yaricap):
5       return PI * yaricap ** 2
6   def daire_cevresi(yaricap):
7       return 2 * PI * yaricap
8   def dikdortgen_alani(uzunluk, genislik):
9       return uzunluk * genislik
10  def ucgen_alani(taban, yukseklik):
11      return 0.5 * taban * yukseklik
```

2. Aynı klasörde **ana_program.py** adında yeni bir dosya oluşturulur.

```
1   import geometri
2
3   alan = geometri.daire_alani(5)
4   cevre = geometri.daire_cevresi(5)
5   print(f'Daire alanı  : {alan:.2f}')
6   print(f'Daire çevresi: {cevre:.2f}')
7   d_alan = geometri.dikdortgen_alani(8, 3)
8   print(f'Dikdörtgen alanı: {d_alan}')
```

EKRAN ÇIKTISI

```
Daire alanı : 78.54
Daire çevresi: 31.42
Dikdörtgen alanı: 24
```

SIRA SİZDE

Sıcaklık dönüşümleri yapan fonksiyonların bulunduğu **donusum.py** adında bir modül tasarlayınız ve bu modülü ana programınızda test ediniz.

Celsius'u fahrenheit'a çevirme formülü:
$F=(C\times1.8)+32$

Fahrenheit'ı celsius'a çevirme formülü:
$C=(F-32)/1.8$

41. ÖRNEK UYGULAMA

Amaç: Modüler taş-kâğıt-makas oyunu tasarlamak

```python
# oyun_kurallari.py
SECENEKLER = ["taş", "kağıt", "makas"]
def kazanan_belirle(oyuncu_secimi, bilgisayar_secimi):
    if oyuncu_secimi == bilgisayar_secimi:
        return "berabere"
    kazanma_durumlari = {
        "taş": "makas",
        "kağıt": "taş",
        "makas": "kağıt"
    }
    if kazanma_durumlari[oyuncu_secimi] == bilgisayar_secimi:
        return "oyuncu"
    return "bilgisayar"
def tur_sonucu_yazdir(sonuc, oyuncu_secimi, bilgisayar_secimi):
    print(f"Sizin hamleniz: {oyuncu_secimi.upper()} | Bilgisayarın hamlesi: {bilgisayar_secimi.upper()}")
    if sonuc == "berabere":
        print("Bu tur berabere bitti.")
    elif sonuc == "oyuncu":
        print("Bu turu siz kazandınız.")
    else:
        print("Bu turu bilgisayar kazandı.")
```

```python
# ana_oyun.py - Ana akış ve konsol arayüzü
from random import choice
from time import sleep
import oyun_kurallari
def turnuvayi_baslat():
    oyuncu_puani = 0
    bilgisayar_puani = 0
    hedef_puan = 3
    print("=" * 45)
    print("TAŞ-KAĞIT-MAKAS TURNUVASI (3 Puan)")
    print("=" * 45)
    while oyuncu_puani < hedef_puan and bilgisayar_puani < hedef_puan:
        print(f"Skor Tablosu -> Siz: {oyuncu_puani} | Bilgisayar: {bilgisayar_puani}")
        secim = input("Hamlenizi seçin (taş, kağıt, makas / çıkış için 'q'): ").strip().lower()
        if secim == "q":
            print("\nTurnuvadan çıkış yapıldı.")
            return
        if secim not in oyun_kurallari.SECENEKLER:
            print("Geçersiz hamle! Lütfen 'taş', 'kağıt' veya 'makas' yazınız.")
            continue
        pc_secimi = choice(oyun_kurallari.SECENEKLER)
        print("Hamleler açılıyor...")
        sleep(0.6)
        tur_sonucu = oyun_kurallari.kazanan_belirle(secim, pc_secimi)
        oyun_kurallari.tur_sonucu_yazdir(tur_sonucu, secim, pc_secimi)
        if tur_sonucu == "oyuncu":
            oyuncu_puani += 1
        elif tur_sonucu == "bilgisayar":
            bilgisayar_puani += 1
    print("\n" + "=" * 45)
    if oyuncu_puani == hedef_puan:
        print(f"TEBRİKLER! Turnuvayı {oyuncu_puani}-{bilgisayar_puani} kazandınız!")
    else:
        print(f"Turnuvayı {bilgisayar_puani}-{oyuncu_puani} bilgisayar kazandı!")
    print("=" * 45)
if __name__ == "__main__":
    turnuvayi_baslat()
```

EKRAN ÇIKTISI

Hamlenizi seçin (taş, kağıt, makas / çıkış için ‘q’): kağıt

Sizin hamleniz: KAĞIT | Bilgisayarın hamlesi: TAŞ

Bu turu siz kazandınız.

2. UYGULAMA

Modüler Not Hesaplayıcısı Geliştiriyorum

Süre: 40 dakika

Amaç: Standart kütüphane modülleri ve programcının yazdığı modülü birlikte kullanarak modüler bir not hesaplayıcı geliştirmek

Yönerge: Aşağıdaki adımları takip ederek uygulamayı geliştiriniz.

1. Adım: notlar.py adında bir modül dosyası oluşturunuz. Bu modül aşağıdaki fonksiyonları içermelidir.

- **ortalama_hesapla(notlar):** bir liste alıp ortalamasını döndürür
- **derece_karsiligi(ortalama):** Ortalamaya göre derece karşılığını döndürür.
  - 85,00 - 100: Pekiyi
  - 70,00 - 84,99: İyi
  - 60,00 - 69,99: Orta
  - 50,00 - 59,99: Geçer
  - 0 - 49,99: Geçmez (Başarısız)
- **alinan_belge(ortalama):** Ortalamaya göre belge durumunu döndürür.
  - Ortalama 70 in altındaysa “Belge Yok”
  - 70-85 arasındaysa “Teşekkür Belgesi”
  - 85-100 arasındaysa “Takdir Belgesi”

2. Adım: random modülünü **takma ad** ile içe aktararak 5 öğrenci için rastgele (0-100 arası) not üreten bir liste oluşturunuz.

3. Adım: math modülünden **from ... import** söz dizimiyle yalnızca **ceil** veya **floor** fonksiyonunu içe aktarıp ortalamayı yuvarlamada kullanınız.

4. Adım: ana_program.py dosyasında kendi yazdığınız notlar modülünü içe aktararak ortalamayı, derece karşılığını ve alınan belgeyi hesaplatıp ekrana yazdırınız.

5. Adım: datetime modülü ile raporun altına hesaplamanın yapıldığı tarihi ekleyiniz.

6. Adım: Programı çalıştırıp ekran çıktısını (öğrenci notları, ortalama, derece karşılığı, alınan belge, tarih) gözlemleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

5.1. PROJE	
Konu	Kişisel Matematik Asistanı Geliştiriyorum
Kapsanan Konular	Fonksiyon tanımlama, parametre, return, kapsam, lambda, özyineleme, modüler programlama
Süre	40+40+40 dakika
Amaç	Fonksiyonlar ve modüler programlamayı kapsayıcı bir projede kullanmak
Çalışma Biçimi	Bireysel

Proje Senaryosu

Lise birinci sınıfta okuyan bir öğrenci matematik dersinde sıkça tekrarlanan hesaplamaları anlamakta zorlanır. Faktöriyel, EBOB, üs alma, asal sayı kontrolü, geometrik şekillerle ilgili hesaplamalar buna örnek olarak verilebilir. Bu öğrenciye yardımcı olmak için **menü tabanlı bir matematik asistanı** uygulaması geliştiriniz. Uygulama, **kendi modülünüzü** içeren bir yapıda tasarlanmalıdır.

Yönerge: Aşağıdaki adımları takip ederek projeyi geliştiriniz.

1. Adım: Planlama, projenin algoritmasını yazınız. Hangi fonksiyonun hangi modülde olacağına önceden karar veriniz.

2. Adım: İki dosya yapısı, Editörünüzde aşağıdaki iki dosyayı oluşturunuz.

- **mat_modul.py:** Tüm yardımcı matematik fonksiyonlarını içerecek modül.
- **asistan.py:** Ana programınız. Modülü import ederek kullanır.

3. Adım: mat_modul.py içine aşağıdaki fonksiyonları ekleyiniz.

- kare(x): Sayının karesini döndürür.
- kup(x): Sayının küpünü döndürür.
- us_al(taban, us): Taban^us değerini döndürür.
- faktoriyel(n): özyineleme ile n! Hesaplar.
- fibonacci(n): özyineleme ile n. Fibonacci sayısını döndürür.
- asal_mi(n): n sayısının asal olup olmadığını döndürür.
- ebob(a, b): İki sayının en büyük ortak bölenini bulur.
- ekok(a, b): İki sayının en küçük ortak katını bulur (a*b/ebob ile).
- daire_alan(r): Dairenin alanını döndürür.
- dikdortgen_olcu(u, g): Demet hâlinde (alan, çevre) döndürür.

4. Adım: asistan.py içine menü yapısı kurunuz. Kullanıcı seçim yapana kadar döngü devam etmelidir.

5. Adım: Lambda kullanımı, kullanıcıdan bir sayı listesi alıp aşağıdaki işlemleri **lambda, filter() ve map()** ile yapınız.

- Çift sayıları filtrele.
- Asal sayıları filtrele (mat_modul.asal_mi ile).
- Her sayının karesini al.
- Listeyi büyükten küçüğe sırala (sorted + lambda).

6. Adım: Test: Programınızı tüm 11 seçenek için en az birer kez test ediniz.

7. Adım: Geçersiz giriş kontrolü: Kullanıcı menüde olmayan bir seçenek girerse uyarı veriniz.

8. Adım: Dijital depoya, her iki dosyayı (modül + ana program) yükleyiniz. README.md dosyasında projenin işlevini ve modülün içeriğini açıklayınız. Anlamlı commit mesajları yazınız.

9. Adım: Dijital depo ve Issue açmayla yazdığınız projeyi geliştirebilmek için depo platformunuzun **"Issues" (sorunlar)** bölümüne giderek projenize en az **iki Issue** oluşturunuz. Şu şekilde oluşturulabilirsiniz:

• **Issue #1 İyileştirme önerisi (enhancement):** "Negatif sayı girildiğinde faktöriyel fonksiyonu uyarı vermeli." başlıklı bir Issue açarak hangi koda eklemeniz gerektiğini, hangi durumlarda eksik kaldığını ve nasıl bir çözüm geliştirmek istediğinizi açıklayınız.

• **Issue #2 İyileştirme önerisi:** "Menüye 13. seçenek olarak 'Geçmişteki işlemleri göster.' eklenmeli." başlıklı bir Issue açarak yapılan işlemlerin bir listede saklanıp menüden gösterilmesinin proje için nasıl bir iyileştirme olacağını açıklayınız.

Her bir Issue için **başlık, açıklama** ve uygun bir **etiket** (enhancement, bug, documentation vb.) belirleyiniz. Issue açma, kişisel projelerinizde yapılan işleri kayıt altına almanın ve iyileştirme önerilerinizi takip etmenin profesyonel yoludur.

10. Adım: Issue için çözüm commitiyle açtığınız Issue'lardan en az birine yönelik kodda gerekli iyileştirmeyi yapınız (örneğin faktöriyel fonksiyonuna negatif sayı kontrolü eklemek). Değişikliğinizi commit ederken **commit mesajında Issue numarasına referans veriniz** (ör. "Fixes #1: faktöriyel fonksiyonuna negatif sayı kontrolü eklendi."). Bu sayede hangi commit'in hangi sorunu çözdüğü depo üzerinden izlenebilir.

Değerlendirme

a) Yapacağınız proje çalışması öğretmeniniz tarafından **Proje Derecelendirme Ölçeğinde** yer alan ölçütlere göre değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak derecelendirme ölçeğine ulaşabilirsiniz.

Proje derecelendirme ölçeğine ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

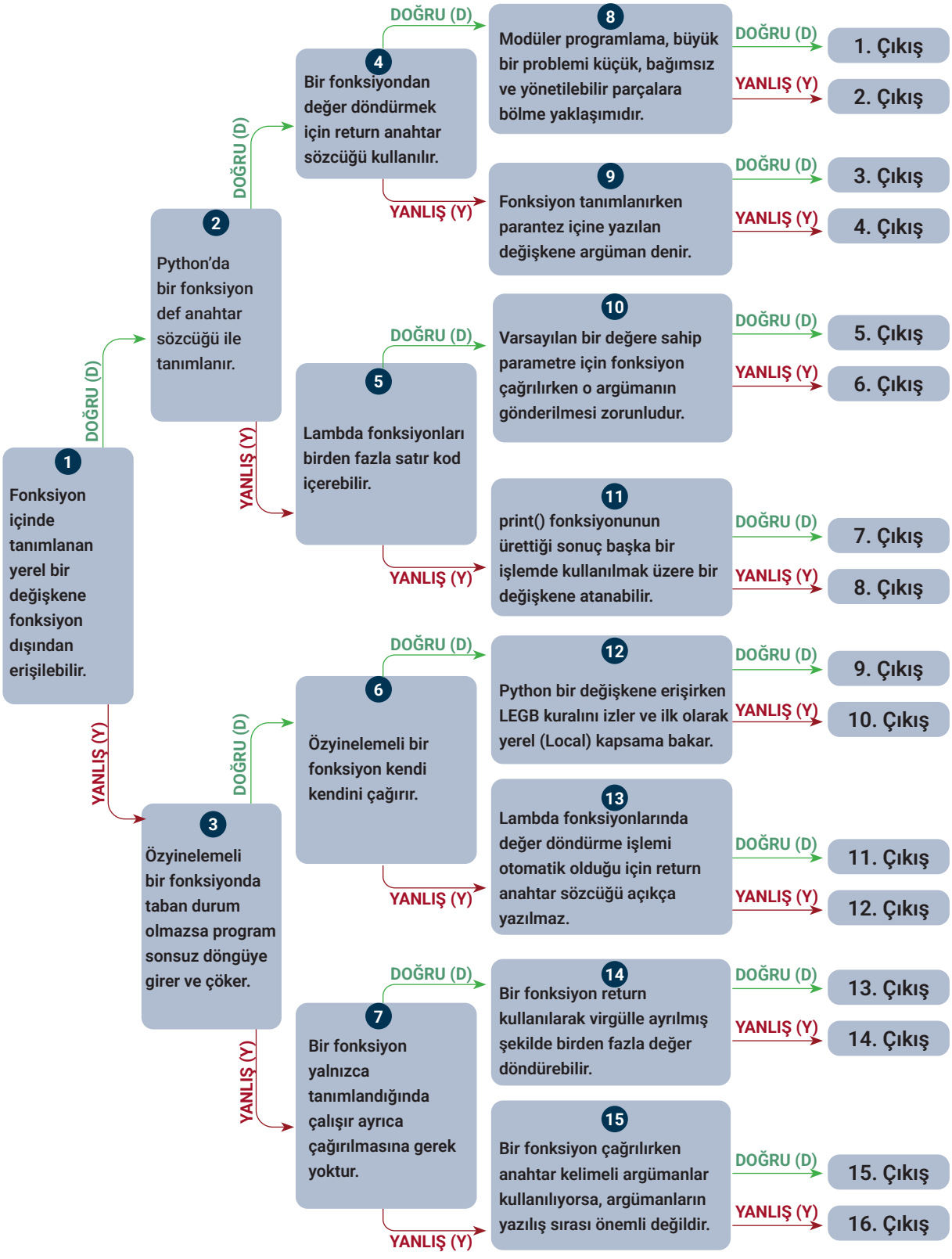
b) **Öz Değerlendirme Formu** proje geliştirme sürecinin sonunda kendinizi değerlendirmeniz amacıyla hazırlanmıştır. Cevaplarınızı tabloda ilgili kutucuğa yazınız. Verilen karekodu kullanarak öz değerlendirme formuna ulaşabilirsiniz.

Öz değerlendirme formuna ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

ÖLÇME VE DEĞERLENDİRME

A) Aşağıda Doğru / Yanlış önermeleri içeren tanılayıcı dallanmış ağaç verilmiştir. Verilen önermenin doğru olduğunu düşünüyorsanız (D), yanlış olduğunu düşünüyorsanız (Y) seçeneğini işaretleyiniz. İlk önermeden başlayarak yapacağınız seçimlerle ve oklar yönünde ilerleyerek bir çıkış noktası belirleyiniz.



B) Aşağıdaki cümlelerde boş bırakılan yerleri doğru kelimelerle doldurunuz.

1. Belirli bir işi yapmak üzere yazılan, ada sahip ve tekrar çağrılabilen kod bloğuna ..... denir.

2. Python'da bir fonksiyon ..... anahtar sözcüğü ile tanımlanır.

3. Fonksiyon tanımlanırken parantez içine yazılan değişkene ..... denir.

4. Fonksiyonun ürettiği sonucu geri döndürmek için ..... anahtar sözcüğü kullanılır.

5. Bir fonksiyonun içinde tanımlanan ve yalnızca o fonksiyonda geçerli olan değişkene ..... değişken denir.

6. Tek satırlık, isimsiz fonksiyonlara ..... fonksiyonu denir.

7. Bir fonksiyonun kendi kendini çağırmasına ..... denir.

8. Özyinelemenin durduğu en basit duruma ..... denir.

9. Büyük bir programı küçük ve bağımsız parçalara bölme yaklaşımına ..... programlama denir.

10. 4! (dört faktöriyel) işleminin sonucu ..... değerine eşittir.

C) Sol sütundaki açıklamayı, sağ sütundaki uygun ifadeye ait harfle eşleştiriniz.

	Açıklama
( )	1. Fonksiyonun kendi kendini çağırması
( )	2. Fonksiyon tanımlanırken kullanılan değişken
( )	3. Tek satırlık, isimsiz fonksiyon
( )	4. Sonucu geri döndüren anahtar sözcük
( )	5. Yalnızca bir fonksiyon içinde geçerli değişken

İfade
a) Parametre
b) return
c) Yerel değişken
ç) Özyineleme
d) Lambda
e) Argüman

Ç) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Python'da bir fonksiyon tanımlamak için hangi anahtar sözcük kullanılır?

A) def
B) func
C) function
D) lambda
E) return

2.

```
def topla(a, b):
    return a + b
print(topla(3, 7))
```

Yukarıdaki kodun çıktısı nedir?

A) 10
B) 37
C) a + b
D) None
E) Hata

3. Fonksiyon çağırılırken parantez içine yazılan gerçek değere ne ad verilir?

A) Parametre
B) Gövde
C) Kapsam
D) Lambda
E) Argüman

4. return anahtar sözcüğü ile ilgili aşağıdakilerden hangisi doğrudur?

A) Fonksiyonu yeniden başlatır.
B) Sonucu ekrana yazar.
C) Sonucu geri döndürür ve fonksiyonu sonlandırır.
D) Yeni bir fonksiyon tanımlar.
E) Hiçbir işlevi yoktur.

5.
```
f = lambda x: x * 3
```
Yukarıdaki lambda fonksiyonunun yaptığı iş nedir?

A) Sayının karesini alır.
B) Sayıyı üçe böler.
C) Sayıya üç ekler.
D) Sayıyı üç ile çarpar.
E) Hata verir.

6. Bir fonksiyon içinde tanımlanan yerel bir değişkene fonksiyon dışından erişilmeye çalışılırsa ne olur?

A) Değişkenin değeri yazılır.
B) 0 değeri yazılır.
C) None yazılır.
D) Program durur ama hata vermez.
E) NameError hatası oluşur.

7. Özyinelemeli bir fonksiyonda taban durum ne işe yarar?

A) Fonksiyonu hızlandırır.
B) Özyinelemenin durmasını sağlar.
C) Fonksiyona yeni parametre ekler.
D) Sonucu ekrana yazar.
E) Fonksiyonu siler.

8.

```
def faktoriyel(n):
    if n == 1:
        return 1
    return n * faktoriyel(n - 1)
print(faktoriyel(3))
```

Yukarıdaki kodun çıktısı nedir?

A) Hata
B) 1
C) 3
D) 6
E) 9

9. Özyinelemeli bir fonksiyonda taban durum unutulursa ne olur?

A) Fonksiyon daha hızlı çalışır.
B) Fonksiyon hiç çalışmaz.
C) Fonksiyon kendini sonsuza kadar çağırır ve program çöker.
D) Fonksiyon 0 döndürür.
E) Hiçbir şey olmaz.

10. Büyük bir programı küçük, bağımsız ve tekrar kullanılabilir parçalara bölmenin sağladığı temel avantaj aşağıdakilerden hangisidir?

A) Program daha yavaş çalışır.
B) Kod tekrarı önlenir, bakım ve okunabilirlik kolaylaşır.
C) Program daha fazla bellek kullanır.
D) Hata sayısı artar.
E) Programın boyutu küçülür.

D) Bu öğrenme biriminde edindiklerinizi değerlendirmek için aşağıdaki çıkış kartını doldurunuz.

Üç Yaz	Fonksiyonlar ve Modüler Programlama öğrenme biriminde öğrendiğiniz üç bilgiyi yazınız.
1	
2	
3	
İki Sor	Fonksiyonlar veya özyineleme ile ilgili merak ettiğiniz iki soruyu yazınız.
1	
2	
Bir Paylaş	Bu öğrenme biriminde geliştirdiğiniz uygulamalardan birini sınıf arkadaşlarınızla paylaşmak isteseniz hangisini seçerdiniz? Neden?



KONULAR

6.1. İSTİSNA İŞLEMLERİ

6.2. HATA AYIKLAMA BLOKLARI

ANAHTAR KAVRAMLAR

- hata ayıklama
- hata ayıklama bloğu
- istisna
- istisna işlemleri
- istisna türleri
- try-except



6. ÖĞRENME BİRİMİ

HATA YAKALAMA İŞLEMLERİ

BEKLENTİLER

Bu öğrenme biriminde sizden

- istisna işlemlerini kullanarak hata yönetimini yapmanız
- hata ayıklama bloklarını kullanmanız

beklenmektedir.

HAZIRLIK ÇALIŞMALARI

1. Aşağıda araç kullanımı ile ilgili bir metin verilmiştir.

"Otomobil kullanırken yakıt seviyesi azaldığında araç aniden yolda kalmaz. Bunun yerine gösterge panelinde turuncu bir ikaz ışığı yanar ve sürücüyü yakıtın azaldığı konusunda uyarır. Sürücü bu uyarıyı dikkate alarak en yakın akaryakıt istasyonuna gider ve yola güvenle devam eder. Otomobilde böyle bir ikaz sistemi olmasaydı aracın yakıtı bittiği anda belki de tehlikeli bir noktada hiçbir uyarı vermeden aniden stop ederdi. Bu durum sürücünün ve yolcuların güvenlikleri açısından büyük risk oluşturabilirdi."

Verilen metne göre aşağıdaki soruları cevaplayınız.

a) İkaz ışığının yanmasının sürücüye sağladığı temel fayda nedir?

b) İkaz ışığı, sorun oluşmadan önce sürücüyü bilgilendirdiği için hangi olumsuz durumların önüne geçer?

2. Daha önce yazdığınız programlarda kullanıcıdan veri almak için hangi komutları kullandınız?

3. Kullanıcıdan alınan verinin sayı mı yoksa metin mi olduğunu kontrol etmek neden gerekli olabilir?

BAŞLARKEN

İyi tasarlanmış sistemler beklenmedik durumlarda aniden durmaz, **kullanıcıyı uyarır** ve mümkünse çözüm yolu sunar. Bir arabanın yakıtı azaldığında ikaz ışığı yakması gibi iyi yazılmış bir program da hata durumlarında kullanıcıyı bilgilendirmelidir.

Programlar kullanıcı **beklendiği gibi** davrandığı sürece sorunsuz çalışır. Kullanıcı yaşının girilmesi istendiğinde "on beş" yazarsa ya da bir sayı 0'a bölünürse program çöker ve kullanıcının karşısına anlaşılması güç bir hata yığını çıkar.

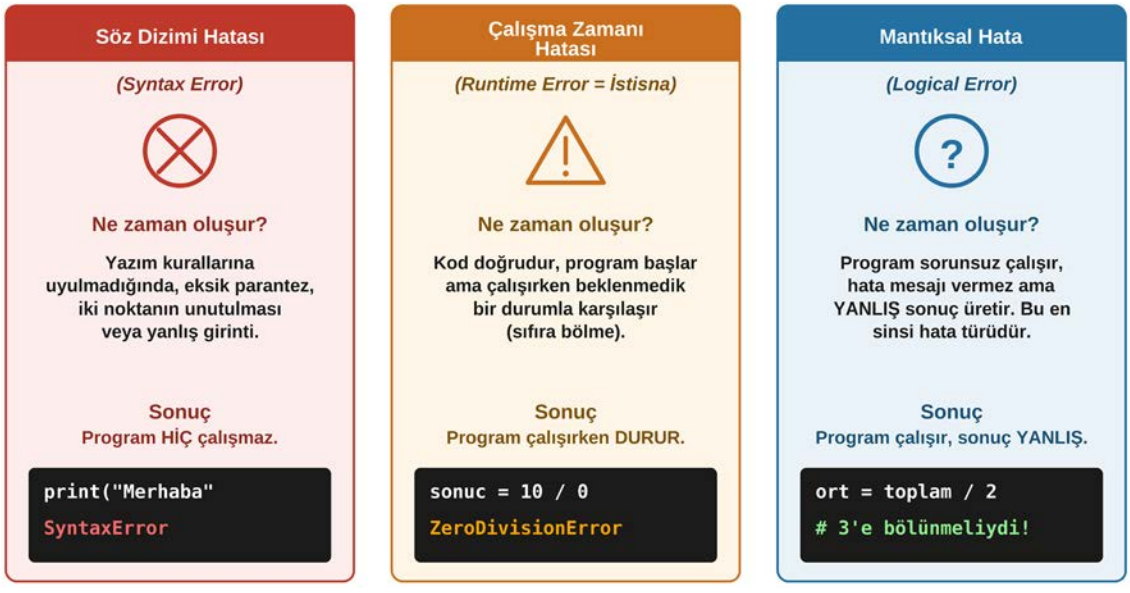
6.1. İSTİSNA İŞLEMLERİ

Program yazıldıktan sonra her zaman beklenildiği gibi çalışmayabilir. Kullanıcı yanlış bir veri girebilir, dosya bulunamayabilir veya hesaplanması mümkün olmayan bir işlem içerebilir. Bu gibi durumlarda program çöker ve kullanıcının karşısına anlaşılması güç hata mesajları çıkar. Beklenmedik durumlarda programın doğru biçimde yönetilmesi gerekir.

Hata Kavramı

Bir programın beklenen şekilde çalışmasını engelleyen her türlü duruma **hata** denir ancak her hata aynı değildir. Bazı hatalar program çalışmaya başlamadan ortaya çıkar, bazıları program çalışırken belirir ve bazıları ise fark edilmeden yanlış sonuçlar üretir. Bu nedenle hataları doğru tanımak, onları çözmenin ilk adımıdır.

Program yazarken karşılaşılan hatalar; **söz dizimi ve çalışma zamanı hataları ile mantıksal hatala**r olmak üzere üç ana grupta toplanır (Görsel 6.1).



Görsel 6.1: Programlamadaki üç temel hata türü

Söz Dizimi Hatası

Programlama dilinin **yazım kurallarına** uyulmadığında söz dizimi hatası (syntax error) ortaya çıkar. Türkçedeki yazım ve noktalama kuralları gibi her programlama dilinin de kendine özgü bir söz dizimi vardır. Derleyici dillerin aksine Python bir yorumlayıcı dili olduğu için kod satır satır okunur. Hata, dosyanın en sonundaysa Python hatadan önceki kısımları sorunsuz çalıştırır ancak hatalı satırda program hata vererek tamamen durur.

1. ÖRNEK UYGULAMA

Amaç: Söz dizimi hatası içeren kodu çalıştırıp sonucunu gözlemlemek

```
1   if (5 > 3)
2       print("Beş, üçten büyüktür")
```

EKRAN ÇIKTISI

```
File "C:\kodlar\6\6_1.py", line 1
  if (5 > 3)
       ^
SyntaxError: expected ':'
```

Yukarıdaki kodda **if** satırının sonunda iki nokta (:) eksiktir. Python hatayı kodu çalıştırmadan önce fark eder ve **SyntaxError** mesajı ile hangi satırda sorun olduğunu bildirir. Hata düzeltilmeden program çalışmaz.

Söz dizimi hataları, deneyim kazandıkça daha hızlı fark edilir. Python'un verdiği hata mesajları genellikle hatanın hangi satırda olduğunu ve ne tür bir sorun bulunduğunu açıkça belirtir. Tablo 6.1'de en sık yapılan söz dizimi hataları, hatalı ve doğru yazımlarıyla birlikte verilmiştir.

Tablo 6.1: Kodlarda Söz Dizimi Hataları

Hata Türü	Hatalı Yazım	Doğru Yazım
Eksik iki nokta	for i in range(5)	for i in range(5):
Kapatılmayan parantez	print("Merhaba"	print("Merhaba")
Kapatılmayan tırnak	ad = "Ayşe	ad = "Ayşe"
Yanlış girinti	if x>0: print(x)	if x>0: print(x)

1. UYGULAMA

Söz Dizimi (Syntax) Hatalarını Tespit Edip Düzeltiyorum

Süre: 40 dakika

Amaç: Python kodundaki yazım hatalarını düzeltmek

Yönerge: Aşağıdaki adımları takip ederek hatalı kodları inceleyiniz ve düzeltiniz.

1. Adım: Editörünüzde **yazim_hatalari.py** adında yeni bir dosya oluşturunuz.

2. Adım: Aşağıdaki **yazım hatası içeren üç kodu** dosyaya kopyalayınız.

```
# Kod 1
print(“Merhaba Dünya”

# Kod 2
sayi = 10
if sayi > 5
   print(“Sayı büyük”)

# Kod 3
for i in range(5):
print(i)
```

3. Adım: Her kod parçasını ayrı ayrı çalıştırınız ve aldığınız hata mesajlarını gözlemleyerek aşağıdaki tabloyu doldurunuz.

Kod No	Aldığı Hata Mesajı	Yazım Hatasının Nedeni	Yapılan Düzeltme
Kod 1			
Kod 2			
Kod 3			

4. Adım: Her yazım hatasını düzelterek kodu yeniden çalıştırınız ve doğru çıktının alındığından emin olunuz.

5. Adım: Aşağıda içinde **birden çok yazım hatası** bulunan daha uzun bir kod verilmiştir. Bu koddaki tüm söz dizimi hatalarını (eksik iki nokta, hatalı girinti, kapatılmayan tırnak ve parantez) bularak düzeltiniz.

```
def selamla(isim)
   mesaj = “Merhaba “ + isim
    print(mesaj)

selamla(“Ayşe)
```

6. Adım: Düzelttiğiniz kodu çalıştırarak “Merhaba Ayşe” çıktısını aldığınızı doğrulayınız.

7. Adım: Düzeltilen tüm kodları dijital deponuza yükleyiniz. README dosyasında düzelttiğiniz her yazım hatasını kısaca açıklayınız.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.



Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



Çalışma Zamanı Hatası

Kod söz dizimi açısından doğru ve program çalışmaya başladığı sırada beklenmedik bir durumla karşılaşıp durursa bu hataya **çalışma zamanı hatası** (runtime error) denir. Bir sayının sıfıra bölünmesi, var olmayan bir dosyanın açılmaya çalışılması, kullanıcının sayı yerine harf girmesi gibi durumlar çalışma zamanı hatasına örnektir. Programlamada **istisna** (exception) kavramı bu çalışma zamanı hatalarını ifade eder.

Mantıksal Hata

Programın **sorunsuz çalıştığı** ve hiçbir hata mesajı vermediği ancak **yanlış sonuç** ürettiği hata türüne **mantıksal hata** (logical error) denir. Bu durum hataların en beklenmeyen türüdür. Programın çöktüğü anlaşılmaz. Hata yalnızca sonuçların yanlış olduğu fark edildiğinde anlaşılır.

İstisna Kavramı

Program çalışırken ortaya çıkan ve programın akışını kesen beklenmedik olaya **istisna** (exception) denir. Söz dizimi hataları program çalışmadan önce ortaya çıkar, istisnalar ise program çalışırken oluşur. Mantıksal hatalarsa hiç istisna üretmez yalnızca yanlış sonuçlara yol açar.

Günlük hayatta istisna durumlar sık sık ortaya çıkar. Asansörün taşıma kapasitesinin aşılması, bankamatikten bakiyeden fazla para çekilmesi veya bir uçağın kötü hava koşulları nedeniyle iniş yapamaması birer istisna durumudur. İyi tasarlanmış sistemler bu durumlarda **çökmek yerine** bir uyarı verir ve güvenli bir şekilde durumu yönetir. Programlamadaki istisna işlemleri de tam olarak bunu yapar. Beklenmedik

durumu fark eder, kullanıcıyı bilgilendirir ve programın kontrollü biçimde çalışmaya devam etmesini sağlar. Tablo 6.2'de hata türleri karşılaştırılmaktadır.

Tablo 6.2: Üç Hata Türünün Karşılaştırması

Karşılaştırma Ölçütleri	Söz Dizimi Hatası	Çalışma Zamanı Hatası (İstisna)	Mantıksal Hata
Ne zaman fark edilir?	Yorumlama (parsing) aşamasında veya kod yazılırken	Program çalışıp tam olarak hatalı satıra gelindiğinde	Programın ürettiği sonuçlar incelendiğinde
Program çalışır mı?	Hayır, Python kodu başlatmayı reddeder.	Evet, hatalı satıra kadar çalışır; hata yakalanmazsa çöker.	Evet, program başından sonuna kadar sorunsuz çalışır.
Hata mesajı verir mi?	Evet (Hatanın yeri ok işareti ile gösterilir).	Evet (Hata izleme raporu sunulur).	Hayır, sistem herhangi bir uyarı vermez.
Fark edilmesi	Çok Kolay	Kolay / Orta	Zor
Tipik örnek	if x == 5 (Eksik iki nokta)	print(10 / 0) (Sıfıra bölme)	alan = pi * r (Kare almayı unutmak)

BİLGİ KUTUSU

Bir programın hata mesajı vermemesi doğru çalıştığı anlamına gelmez. Mantıksal hatalar program tarafından fark edilemez. Bu nedenle yazılan programlar her zaman farklı verilerle test edilmeli ve sonuçların beklendiği gibi olup olmadığını kontrol edilmelidir.

2. ÖRNEK UYGULAMA

Amaç: Aynı kodun farklı verilerle gösterdiği hata davranışlarını gözlemlemek

Aşağıdaki kodla kullanıcıdan alınan iki sayıyı bölünür. Aynı kod, kullanıcının girdiği değere göre bazen sorunsuz çalışır bazen de istisna fırlatılır.

```
1   a = int(input("Birinci sayı: "))
2   b = int(input("İkinci sayı: "))
3   print("Bölüm:", a / b)
```

Bu kodun farklı değerlerle çalıştırılması ve sonuçları Tablo 6.3'te verilmiştir.

Tablo 6.3: Uygulamanın Farklı Değerlerle Test Edilmesi

Girilen Değerler	Sonuç	Açıklama
a=10, b=2	Bölüm: 5.0	Sorunsuz çalışır, hata yoktur.
a=10, b=0	ZeroDivisionError	Sıfıra bölme istisnası oluşur.
a=on, b=2	ValueError	Sayı yerine harf girildiği için istisna oluşur.

Kodun hatasız çalışması yalnızca koda değil kullanıcının girdiği verilere de bağlıdır. İyi bir programcı tarafından kullanıcının yapabileceği tüm hatalı girişler önceden düşünülmelidir.

SIRA SİZDE

Aşağıdaki kod parçalarını inceleyiniz. Her birinin hangi hata türüne (söz dizimi, çalışma zamanı, mantıksal) ait olduğunu gerekçesiyle birlikte belirleyiniz.

1.
```
print("Merhaba"
```

2.
```
liste = [1, 2, 3]
print(liste[10])
```

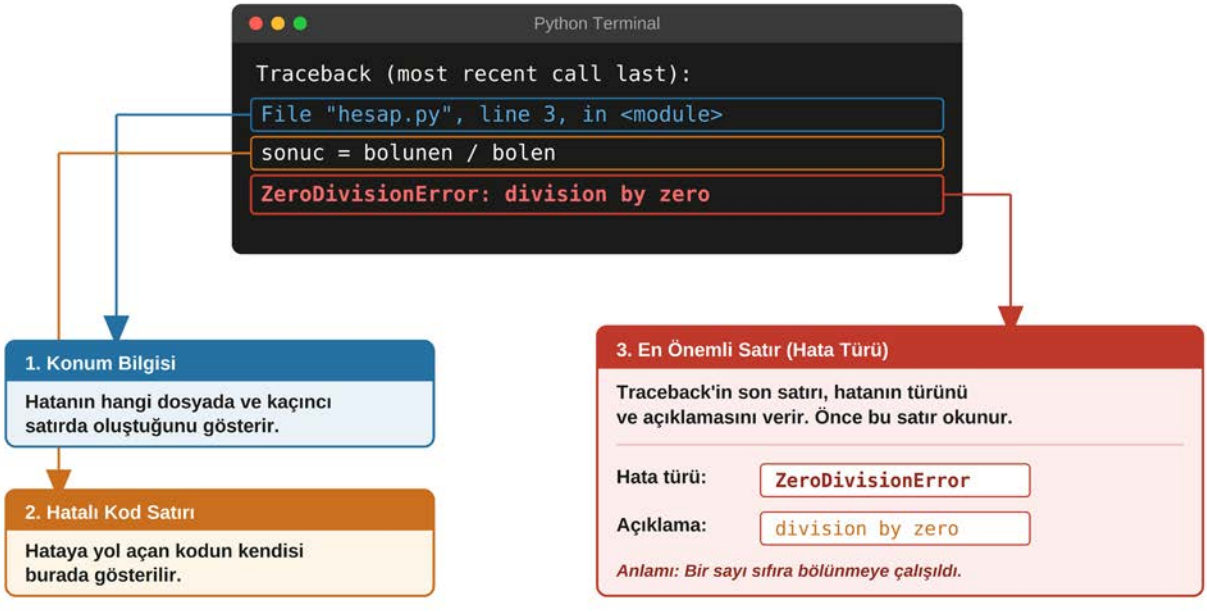
3.
```
kdv_orani = 18
fiyat = 100
kdvli_fiyat = fiyat + kdv_orani
```

4. Kod parçasında KDV'li fiyat yanlış hesaplanır. Doğru hesaplama nasıl olmalıdır?

İstisna Türleri

Python'da her istisnanın bir adı (türü) vardır. Bu ad, hatanın ne tür bir sorundan kaynaklandığını belirtir. Bir programcının hangi durumda hangi istisnanın oluştuğunu bilmesi, ileride bu hataları doğru biçimde yakalayıp yönetebilmesi için son derece önemlidir.

İstisna oluştuğunda Python ekrana **Traceback** adı verilen bir hata raporu yazdırır. Bu raporun nasıl okunacağı Görsel 6.2'de gösterilmiştir.



Görsel 6.2: Hata raporunun okunması

Traceback raporunun **en önemli satırı sonuncusudur.** Bu satır, oluşan istisnanın türünü (ZeroDivisionError) ve kısa açıklamasını (division by zero) verir. Bir hatanın çözülmesine her zaman bu son satır okunarak başlanır.

Tablo 6.4'te Python'da en sık karşılaşılan istisna türleri, ne zaman oluştukları birer örnekle gösterilmiştir.

Tablo 6.4: Python'da Yaygın İstisna Türleri

İstisna Türü	Oluşma Zamanı	Örnek Kod
ValueError	Bir fonksiyona doğru türde ama uygunsuz değerli bir argüman verildiğinde	int("abc")
ZeroDivisionError	Bir sayı sıfıra bölündüğünde	10 / 0
TypeError	Bir işlem, birbiriyle uyumsuz veri türlerine uygulandığında	"5" + 3
IndexError	Bir listede var olmayan bir indekse erişilmeye çalışıldığında	liste[99]
KeyError	Bir sözlükte var olmayan bir anahtara erişilmeye çalışıldığında	sozluk["yok"]
NameError	Tanımlanmamış bir değişken kullanıldığında	print(x)
FileNotFoundError	Var olmayan bir dosya açılmaya çalışıldığında	open("yok.txt")
AttributeError	Bir nesnenin sahip olmadığı bir özellik çağrıldığında	"metin".artir()

3. ÖRNEK UYGULAMA

Amaç: Farklı kod parçalarının fırlattığı istisna türlerini gözlemlemek

```
# Her satır ayrı ayrı çalıştırılıp fırlattığı istisna gözlemlenir
print(int("on beş"))      # ValueError
print([1, 2, 3][8])       # IndexError
print({"ad": "Ece"}["yas"])  # KeyError
print(toplam)             # NameError
```

EKRAN ÇIKTISI

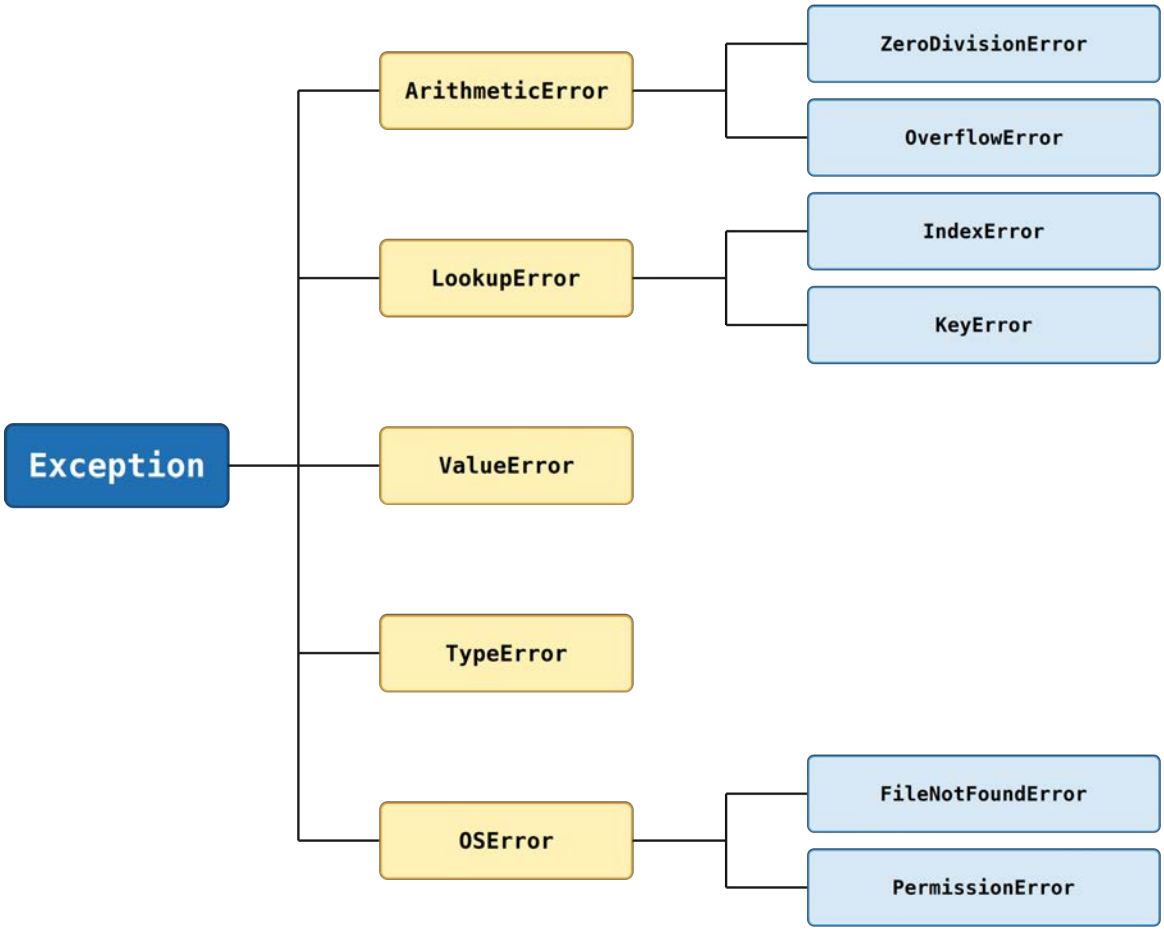
```
Traceback (most recent call last):
  File "C:\kodlar\6\6_6.py", line 2, in <module>
    print(int("on beş"))      # ValueError
          ~~~^^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'on beş'
```

Örnekteki her satırda farklı bir istisna türü fırlatılır. Her istisnanın adı, hatanın kaynağı hakkında ipucu verilir: ValueError değerle ilgili, IndexError liste indeksiyle ilgili; KeyError ise sözlük anahtarıyla ilgili sorunun işaret edildiği görülür.

BİLGİ KUTUSU

Python'da tüm istisna türleri **Exception** adı verilen ortak bir ata sınıftan türetilir. ValueError, TypeError, KeyError gibi hataların hepsi aslında birer Exception'dır. İstisnalar arasında bir hiyerarşi (aile ağacı) vardır.

İstisna türleri arasında aile ağacına benzeyen bir **hiyerarşi** bulunur. En tepede tüm istisnaların atası olan **BaseException** vardır. Günlük programlamada **Exception** sınıfı yer alır. Onun altında daha özel istisna türleri bulunur. Bu hiyerarşi Görsel 6.3'te gösterilmiştir.



Görsel 6.3: Python istisna türleri hiyerarşisi

4. ÖRNEK UYGULAMA

Amaç: Bir istisnanın türünü type() fonksiyonu ile öğrenmek

```
try:
    sonuc = 10 / 0
except Exception as hata:
    print("Oluşan istisnanın türü:", type(hata).__name__)
    print("Hata mesajı:", hata)
```

EKRAN ÇIKTISI

```
Oluşan istisnanın türü: ZeroDivisionError
Hata mesajı: division by zero
```

Bu örnekte **except Exception as hata** yazımı her türlü istisnayı yakalar. Yakalanan istisnanın türü **type(hata).__name__** ile öğrenilebilir. Bu, hangi hatanın oluştuğunu önceden bilinmeyen durumlarda yararlıdır.

5. ÖRNEK UYGULAMA

Amaç: Farklı istisna türlerinin aynı program parçasında oluşmasını gözlemlemek

Aşağıdaki programda bir sözlükten öğrenci notu okunmaktadır. Kullanıcının girdiği veriye göre farklı istisna türleri oluşabilir.

```
1   notlar = {"Ali": 85, "Veli": 90, "Ece": 78}
2   isim = input("Öğrenci adı: ")
3   print("Not:", notlar[isim])
```

Bu kodun farklı değerlerle çalıştırılması ve sonuçları Tablo 6.5'te verilmiştir.

Tablo 6.5: Uygulamanın Farklı Değerlerle Test Edilmesi

Girilen İsim	Sonuç	Açıklama
Ali	Not: 85	İsim sözlükte var, sorunsuz çalışır.
Ahmet	KeyError	İsim sözlükte yok, istisna oluşur.
(boş bırakılır)	KeyError	Boş metin de sözlükte bir anahtar değildir.

SIRA SİZDE

Aşağıda günlük hayatta karşılaşılan bazı durumlar verilmiştir. Her durumu, ona en çok benzeyen Python istisna türüyle ilişkilendiriniz ve neden o türü seçtiğinizi açıklayınız. IndexError, TypeError ve ZeroDivisionError istisna türlerini kullanabilirsiniz.

1. Bir markette olmayan bir ürünü raf numarasıyla aramak
2. Telefonunuzu şarj etmek için USB kablosunu bahçedeki musluğa takmak
3. Bir pastayı sıfır kişiye eşit bölmeye çalışmak
4. Elli sayfalık bir kitabın altmışıncı sayfasını açmaya çalışmak

İstisna Türlerini Tanıma

Programcı için istisna türlerini tanımak, doktorun hastalık belirtilerini tanımasına benzer. Doktor, belirtilere bakarak hastalığı teşhis eder ve doğru tedaviyi uygular. Aynı şekilde programcı da oluşan istisnanın türüne bakarak sorunun kaynağını anlar ve doğru çözümü üretir. Örneğin **ValueError** görüldüğünde sorunun değer dönüşümünde, **KeyError** görüldüğünde ise sözlük erişiminde olduğu anlaşılır. Bu nedenle Traceback raporunun son satırını okumak ve istisna türünü tanımak, hata çözmenin en önemli adımıdır.

Tablo 6.6'da farklı istisna türleri için kullanıcıya gösterilebilecek anlamlı mesaj örnekleri listelenmiştir. İyi bir hata mesajı, kullanıcıya ne yapması gerektiğini söyler.

Tablo 6.6: İstisna Türlerine Göre Uygun Hata Mesajları Hazırlama

İstisna Türü	Kötü Mesaj	İyi Mesaj
ValueError	Hata oluştu!	Lütfen rakamlardan oluşan bir sayı giriniz.
ZeroDivisionError	Error!	Bölme işlemi için sıfırdan farklı bir sayı giriniz.
FileNotFoundError	Dosya hatası	Aradığınız dosya bulunamadı, dosya adını kontrol ediniz.
KeyError	Anahtar yok	Aradığınız bilgi kayıtlarda bulunamadı.

İyi bir hata mesajı kısa, anlaşılır ve yol göstericidir. Kullanıcıya yalnızca "hata var" demek yerine hatanın nasıl giderileceğini açıklar.

SIRA SİZDE

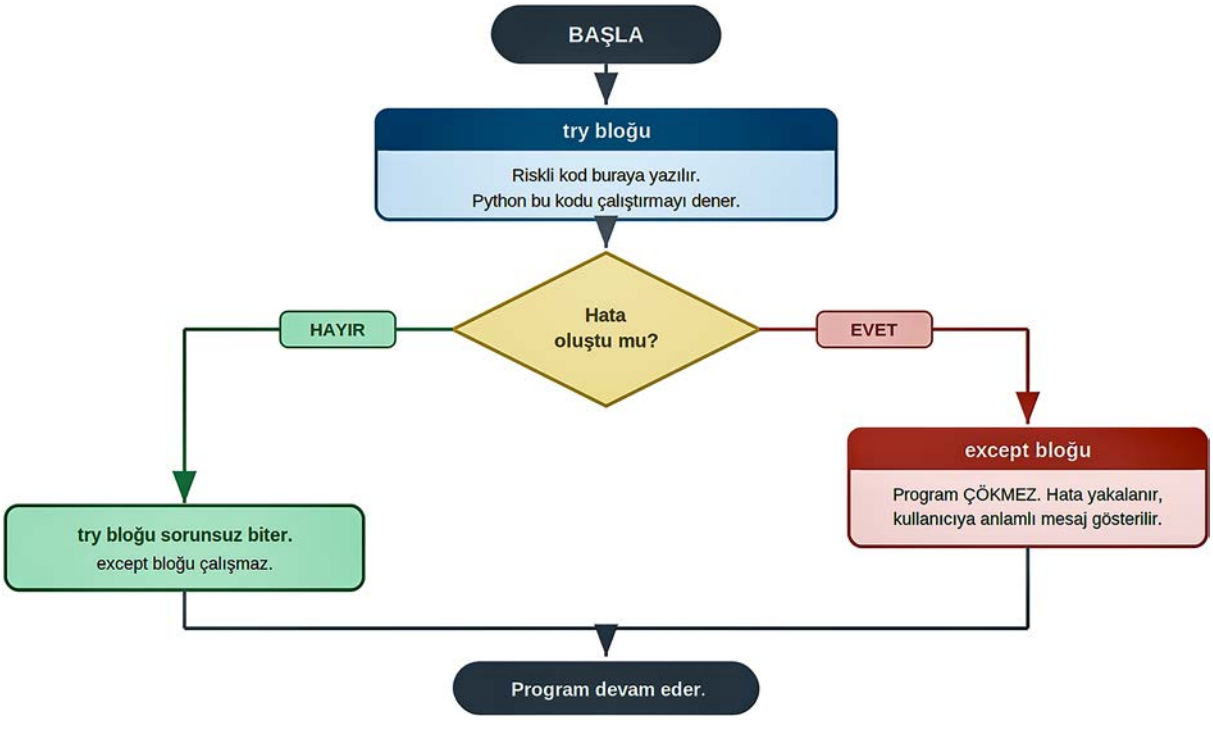
Aşağıdaki istisna türleri için kullanıcıya gösterilebilecek anlamlı, anlaşılır ve kullanıcıya ne yapması gerektiğini söyleyen birer hata mesajı yazınız.

İstisna Türü	Durum	Yazılacak Hata Mesajı
TypeError	Bir metin ile sayı toplanmaya çalışıldığında	
ZeroDivisionError	İki sayıyı bölen bir hesap makinesi uygulamasında ve kullanıcının ikinci sayı (bölen) olarak 0 girmesi girildiğinde	
IndexError	Listede olmayan bir indekse erişildiğinde	

İstisnaları yakalayıp programın çökmesini önlemek için try-except blokları kullanılır.

try-except yapısının mantığı günlük hayat örneğiyle açıklanabilir. Bir bardak sıcak çay taşırken "Ya elimden kayarsa!" diye düşünülerek altına bir tepsi tutulur. Bardak kaymazsa tepsi hiç işe yaramaz. Kayarsa tepsi onu yakalar ve zarar önlenir. İşte **try** bloğu "riskli işlem", **except** bloğu ise "altına tutulan tepsi" gibidir.

Riskli kod **try** bloğunun içine yazılır. Eğer bu kod bir istisna fırlatırsa, program çökmek yerine **except** bloğuna geçer ve oradaki kodu çalıştırır. Eğer hata oluşmazsa except bloğu hiç çalışmaz. Bu akış Görsel 6.4'te gösterilmiştir.



Görsel 6.4: try-except bloğunun çalışma mantığı

try-except yapısının genel söz dizimi şu şekildedir:

```
try:
    # Hata fırlatma ihtimali olan riskli kod
    riskli_islem
except HataTuru:
    # İstisna oluşursa çalışacak kod
    hata_durumunda_yapilacak_islem
```

BİLGİ KUTUSU

try ve except anahtar kelimelerinden sonra gelen kod blokları da tıpkı koşul (if) ve döngü (for/while) yapılarında olduğu gibi girintili (içeriden) yazılmalıdır. Bu kurala uyulmaması durumunda Python, girinti hatası (IndentationError) verir.

6. ÖRNEK UYGULAMA

Amaç: try-except ile sıfıra bölme istisnasını yakalamak

```
try:
    sonuc = 10 / 0
    print("Sonuç:", sonuc)
except ZeroDivisionError:
    print("Hata: Bir sayı sıfıra bölünemez!")

print("Program çalışmaya devam ediyor.")
```

EKRAN ÇIKTISI

```
Hata: Bir sayı sıfıra bölünemez!
Program çalışmaya devam ediyor.
```

try bloğundaki bölme işlemiyle bir istisna fırlatılır ve programla except bloğuna geçilir. Kullanıcıya anlamlı bir mesaj gösterilir ve ardından son satır çalıştırılarak normal şekilde sona erer. Eğer try-except kullanılmazsa program çöker ve "Program çalışmaya devam ediyor" satırı hiç çalışmaz.

BİLGİ KUTUSU

try bloğunun içine sadece istisna fırlatma ihtimali olan "riskli" satırlar yazılmalıdır. Bütün kodları geniş bir try bloğunun içine almak başlangıçta kolay bir çözüm gibi görünebilir. Bu durum riskli olmayan satırlarda yapılabilecek basit hataların (örneğin bir değişken adının yanlış yazılması) yanlışlıkla except bloğuna düşmesine neden olur. Hatanın asıl kaynağının bulunması zorlaşır. İyi tasarlanmış programlarda riskli işlemler olabildiğince dar bir alanda izole edilir. Böylece kodda okunulabilirlik ve güvenilirlik artar.

7. ÖRNEK UYGULAMA

Amaç: Kullanıcı girişinde oluşabilecek ValueError istisnasını yakalamak

```
try:
    yas = int(input("Yaşınızı giriniz: "))
    print("Gelecek yıl", yas + 1, "yaşında olacaksınız.")
except ValueError:
    print("Hata: Lütfen geçerli bir sayı giriniz!")
```

EKRAN ÇIKTISI

```
Yaşınızı giriniz: yirmi
Hata: Lütfen geçerli bir sayı giriniz!
```

Kullanıcı sayı yerine "yirmi" yazdığında int() fonksiyonuyla bir ValueError fırlatılır. except bloğuyla bu hata yakalanır ve kullanıcıya uyarı gösterilir.

8. ÖRNEK UYGULAMA

Amaç: Bir liste işleminde IndexError istisnasını yakalamak

try-except yapısı yalnızca kullanıcı girişiyle değil, her türlü riskli işlemle kullanılabilir. Örnekte listeden eleman okurken oluşabilecek IndexError istisnası yakalanır.

```
ogrenciler = ["Ayşe", "Mehmet", "Zeynep"]
sira = 5

try:
    print(sira, "numaralı öğrenci:", ogrenciler[sira])
except IndexError:
    print("Hata: Listede", sira, "numaralı öğrenci bulunmuyor.")
    print("Listede toplam", len(ogrenciler), "öğrenci var.")
```

EKRAN ÇIKTISI

```
Hata: Listede 5 numaralı öğrenci bulunmuyor.
Listede toplam 3 öğrenci var.
```

Liste yalnızca üç öğrenci içerdiği için 5. indekse erişmek **IndexError** fırlatır. except bloğuyla bu hata yakalanarak kullanıcıya sorunun nedeni ve listenin gerçek boyutu bildirilir. Bu mesajla kullanıcının hatayı anlaması kolaylaştırılır.

2. UYGULAMA

Sözlüklerde ve Değişkenlerde Boş(None) Veri Yönetimi Yapıyorum

Süre: 40 dakika

Amaç: Null veya boş değerlerin kodda neden olduğu hataları izlemek

Yönerge: Aşağıdaki adımları takip ederek null ve boş değer hatalarını izleyiniz.

1. Adım: Editörünüzde **bos_deger.py** adında yeni bir dosya oluşturunuz.
2. Adım: Dördüncü öğrenme biriminden hatırlayacağınız bir **öğrenci sözlüğü** tanımlayınız.

```
ogrenci = {“ad”: “Ayşe”, “yas”: 15, “sinif”: “9-A”}
```

3. Adım: Sözlükte **boş kalan (var olmayan) bir anahtara** erişmeyi try-except KeyError ile izleyiniz.
4. Adım: Aynı sonucu **dict.get()** metodu ile daha güvenli biçimde elde ediniz. get() metodu, olmayan anahtar için istisna fırlatmak yerine varsayılan bir değer döndürünüz.
5. Adım: N**one (null) değeri kontrolü** yapınız. None, “değer yok” anlamına gelen özel bir veridir; sayı, metin gibi işlenemez.
6. Adım: None değerini sayısal bir işlemde kullanmaya çalışınca ne olduğunu try-except ile gözlemleyiniz.
7. Adım: Boş bir listeden eleman almayı (IndexError) ve kullanıcının hiçbir şey yazmadan Enter’a basması (boş metin) durumunu test ederek izleyiniz.
8. Adım: Programınızı README dosyasıyla birlikte dijital deponuza yükleyiniz

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Birden Fazla Except Bloğu

Bir **try** bloğu farklı türde istisnalar fırlatır. Bu durumda her istisna türü için **ayrı bir except bloğu** yazılır. Böylece her hata türüne özel, anlamlı bir mesaj gösterilir.

9. ÖRNEK UYGULAMA

Amaç: Birden fazla except bloğu ile farklı istisnaları ayrı ayrı yönetmek

```
try:
    bolunen = int(input("Bölünen sayıyı giriniz: "))
    bolen = int(input("Bölen sayıyı giriniz: "))
    sonuc = bolunen / bolen
    print("Sonuç:", sonuc)
except ValueError:
    print("Hata: Sayı yerine harf girdiniz!")
except ZeroDivisionError:
    print("Hata: Bölen sıfır olamaz!")
```

EKRAN ÇIKTISI

```
Bölünen sayıyı giriniz: 50
Bölen sayıyı giriniz: 0
Hata: Bölen sıfır olamaz!
```

Bu programda iki farklı hata olasılığı bulunur. Kullanıcı tarafından harf girilebilir (ValueError) veya bölen olarak sıfır girilebilir (ZeroDivisionError). Her iki duruma da ayrı except blokları ile özel mesajlar tanımlanır. Pythonla oluşan istisnanın türüne uygun except bloğu otomatik olarak seçilir.

BİLGİ KUTUSU

Tüm istisnaları tek bir blokta yakalamak için “except Exception as hata:” yazımı kullanılır. Buradaki “as hata” ifadesi, oluşan istisna nesnesini “hata” değişkenine atar ve print(hata) ile hata mesajı ekrana yazdırılır. Ancak mümkün olduğunda hangi hatanın beklendiğini açıkça belirtmek (ValueError, ZeroDivisionError gibi) daha iyi bir programlama alışkanlığıdır.

10. ÖRNEK UYGULAMA

Amaç: Tüm istisnaları tek blokta yakalayıp hata mesajını göstermek

```
try:
    liste = [10, 20, 30]
    print(liste[5])
except Exception as hata:
    print("Bir hata oluştu!")
    print("Hata türü:", type(hata).__name__)
    print("Açıklama:", hata)
```

EKRAN ÇIKTISI
Bir hata oluştu!
Hata türü: IndexError
Açıklama: list index out of range

Birden fazla istisna türüne aynı tepki verilmek istenilebilir. Bu durumda istisna türleri parantez içinde virgülle ayrılarak tek bir except bloğunda toplanır.

11. ÖRNEK UYGULAMA

Amaç: İki farklı istisnayı tek except bloğunda birlikte yakalamak

```
try:
    sayi = int(input("Bir sayı giriniz: "))
    sonuc = 100 / sayi
    print("Sonuç:", sonuc)
except (ValueError, ZeroDivisionError):
    print("Geçersiz bir değer girdiniz, lütfen tekrar deneyiniz.")
```

EKRAN ÇIKTISI
Bir sayı giriniz: 0
Geçersiz bir değer girdiniz, lütfen tekrar deneyiniz.

ValueError ve ZeroDivisionError aynı **except (ValueError, ZeroDivisionError)** bloğunda yakalanır ve her ikisine de aynı mesaj gösterilir. Bu yazım, iki hataya da aynı tepki verilecekse kodu kısaltır.

SIRA SİZDE

Aşağıdaki kod kullanıcıdan bir liste indeksi alır ve o indeksteki şehir bilgisini yazdırır ancak kullanıcı hatalı bir değer girdiğinde program çöker.

```
sehirler = [“Ankara”, “İzmir”, “Bursa”]

indeks = int(input(“İndeks giriniz: “))

print(sehirler[indeks])
```

Bu kodu try-except blokları ile yeniden yazınız. İki ayrı except bloğu kullanınız. Biri sayı yerine harf girilmesi (ValueError) diğeri listede olmayan bir indeks girilmesi (IndexError) durumunu yakalasın. Her duruma uygun bir uyarı mesajı gösteriniz.

try-except yapısı **else** ve **finally** ek bloklarıyla genişletilir. Bu bloklarla hata yönetimini daha esnek hâle getirilir.

else bloğu: try bloğunda **hiç istisna oluşmazsa** çalışır. Hatasız durumda yapılması gereken işler bu bloğa yazılır. Hata oluşursa else bloğu çalışmaz.

finally bloğu: İstisna olsa da olmasa da **her durumda** çalışır. Dosya kapatma, bağlantı sonlandırma gibi her hâlükârda yapılması gereken temizlik işlemleri bu bloğa yazılır.

12. ÖRNEK UYGULAMA

Amaç: else ve finally bloklarının çalışmasını gözlemlemek

```
try:
    sayi = int(input("Bir sayı giriniz: "))
    sonuc = 100 / sayi
except ValueError:
    print("Geçerli bir sayı girmediniz.")
except ZeroDivisionError:
    print("Sıfır giremezsiniz.")
else:
    print("İşlem başarılı! Sonuç:", sonuc)
finally:
    print("İşlem tamamlandı.")
```

EKRAN ÇIKTISI
Bir sayı giriniz: 4
İşlem başarılı! Sonuç: 25.0
İşlem tamamlandı.

Kullanıcıya geçerli bir sayı (4) girildiğinde hatanın oluşmadığı belirtilir. else bloğunun çalışarak sonucun gösterildiği, finally bloğununsa hata olsa da olmasa da çalıştığı ve "İşlem tamamlandı." yazısının her durumda yazıldığı görülür.

Kullanıcı tarafından sıfır girilirse çıktının aşağıdaki gibi olduğu belirtilir.

EKRAN ÇIKTISI
Bir sayı giriniz: 0
Sıfır giremezsiniz.
İşlem tamamlandı.

Bu durumda hata oluştuğu için else bloğunun çalışmadığı, finally bloğunun çalıştırıldığı ifade edilir.

3. UYGULAMA

try-except ile Güvenli Bölme İşlemi Yapıyorum

Süre: 40 dakika

Amaç: Sıfıra bölme işleminde hatayı yakalayarak kullanıcıya anlamlı bir mesaj gösteren uygulama geliştirmek

Yönerge: Aşağıdaki adımları takip ederek uygulamayı geliştiriniz.

1. Adım: Editörünüzde **bolme_islemi.py** adında yeni bir dosya oluşturunuz.

2. Adım: Kullanıcıdan **bölünen ve bölen** sayıyı alacak iki input satırını bir try bloğunun içine yazınız. Sayıları int() ile tam sayıya çeviriniz.

3. Adım: İki ayrı except bloğu yazınız. Birincisi ZeroDivisionError (kullanıcı böleni sıfır girdiğinde), ikincisi ValueError (kullanıcı sayı yerine harf girdiğinde) istisnasını yakalasın. Her birine kullanıcıyı bilgilendiren anlamlı bir mesaj ekleyiniz.

4. Adım: Bölme işlemi başarılıysa sonucu gösteren bir else bloğu ekleyiniz.

5. Adım: Programınızı üç farklı durumla test ediniz: (a) iki geçerli sayı, (b) bölen olarak sıfır, (c) sayı yerine harf. Her durumda programın doğru tepki verdiğini gözlemleyiniz.

Test Senaryosu	1. Sayı	2. Sayı	Beklenen Çıktı
Normal İşlem (Hata Yok)	20	5	4.0
Sıfıra Bölme Hatası	10	0	Hata: Bir sayı sıfıra bölünemez.
Değer Hatası	15	“iki”	Hata: Lütfen sadece rakam giriniz.

6. Adım: Tamamlanan programınızı README dosyasıyla birlikte dijital deponuza yükleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.



Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



raise ile İstisna Fırlatma

Bazen bir değer, programın kuralları açısından kabul edilemez. Örneğin bir yaşın negatif olması veya bir sınav notunun 100’den büyük olması gibi. Bu durumlarda programcı, **raise** anahtar sözcüğünü kullanarak istisnayı **kendisi bilinçli olarak** fırlatır.

raise yapısının genel söz dizimi şu şekildedir:

```
raise IstisnaTuru(“Açıklayıcı hata mesajı”)
```

13. ÖRNEK UYGULAMA

Amaç: raise ile bilinçli olarak istisna fırlatmak

```
1   def yas_dogrula(yas):
2       if yas < 0:
3           raise ValueError("Yaş negatif olamaz!")
4       if yas > 120:
5           raise ValueError("Yaş 120'den büyük olamaz!")
6       return "Yaş geçerli: " + str(yas)
7
8   try:
9       print(yas_dogrula(25))
10      print(yas_dogrula(-7))
11  except ValueError as hata:
12      print("Hata yakalandı:", hata)
```

EKRAN ÇIKTISI

```
Yaş geçerli: 25
Hata yakalandı: Yaş negatif olamaz!
```

Bu örnekte **yas_dogrula** fonksiyonu, geçersiz bir yaş değeri aldığında raise tarafından bir ValueError fırlatılır. Bu istisna, fonksiyonu çağıran try-except bloğuyla yakalanır.

14. ÖRNEK UYGULAMA

Amaç: try-except yapısını bir döngü içinde kullanmak

```
1 while True:
2     try:
3         yas = int(input("Yaşınızı giriniz: "))
4         print("Yaşınız:", yas)
5         break
6     except ValueError:
7         print("Geçersiz giriş! Lütfen bir sayı giriniz.")
```

EKRAN ÇIKTISI

```
Yaşınızı giriniz: abc
Geçersiz giriş! Lütfen bir sayı giriniz.
Yaşınızı giriniz: -
Geçersiz giriş! Lütfen bir sayı giriniz.
Yaşınızı giriniz: 16
Yaşınız: 16
```

while True döngüsü program akışını sürekli tekrarlar. Kullanıcı geçerli bir sayı girdiğinde **break** ile döngüden çıkılır. Geçersiz giriş yapıldığında ise except bloğu çalıştırılır ve döngüyle kullanıcıdan yeni bir giriş istenir. Bu kalıp kullanıcı dostu programlar yazmak için sıkça kullanılır.

4. UYGULAMA

raise İle Değer Doğrulama ve İstisna Fırlatma İşlemlerini Uyguluyorum

Süre: 40 dakika

Amaç: Değişkenlerin beklenmedik bir değer almasını kontrol eden uygulama geliştirmek

Yönerge: Aşağıdaki adımları takip ederek uygulamayı geliştiriniz.

1. Adım: Editörünüzde **deger_kontrolu.py** adında yeni bir dosya oluşturunuz.

2. Adım: Sınav notunu kontrol eden **not_dogrula** adında fonksiyon yazınız. Notun 0-100 aralığında olması beklenir. Bu aralık dışında beklenmedik bir değer girilirse **raise** ile bir ValueError fırlatınız.

3. Adım: Benzer şekilde bir **yas_dogrula** fonksiyonu yazınız, yaş negatif veya 120'den büyük olamaz. Beklenmedik değerlerde raise ile istisna fırlatınız.

4. Adım: Bir de **sicaklik_dogrula** fonksiyonu ekleyiniz, hava sıcaklığı -90 ile +60 derece aralığında olmalıdır (dünya üzerinde ölçülmüş uç değerler). Bu aralık dışındaki değerleri beklenmedik kabul ediniz.

5. Adım: Üç fonksiyonu da **try-except** blokları içinde çağırarak hem geçerli hem de beklenmedik değerlerle test ediniz.

6. Adım: Her fonksiyon için en az bir geçerli ve bir beklenmedik değer test ediniz. Sonuçları tabloda not ediniz.

7. Adım: Programınızı README dosyasıyla birlikte dijital deponuza yükleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.



Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



BİLGİ KUTUSU

Fonksiyonlarda hatalı bir durumla karşılaşıldığında ekrana sadece print() ile uyarı yazdırmak programın çalışmasını durdurmaz. Program hatalı veriyle (örneğin negatif bir yaşla) çalışmaya devam eder ve ileride çok daha büyük sorunlara yol açar. raise kullanmak ise tehlikeyi anında fark edip program akışını keser ve hatayı güvenli bir şekilde except bloğuna gönderir. Profesyonel yazılımlarda hatalar ekrana yazdırılarak değil, istisna fırlatılarak yönetilir.

5. UYGULAMA

Tür Dönüşümü ve Tür Uyuşmazlığı Hatalarını Test Ediyorum

Süre: 40 dakika

Amaç: Veri türleri arasında uyumsuzluktan kaynaklanan hataları belirlemek

Yönerge: Aşağıdaki adımları takip ederek veri türü uyumsuzluğundan doğan hataları belirleyiniz.

1. Adım: Editörünüzde **tur_uyumsuzlugu.py** adında yeni bir dosya oluşturunuz.

2. Adım: Aşağıdaki dört kodu birer birer çalıştırınız ve her birinin hangi istisnayı fırlattığını tabloya yazınız.

```
# Kod 1
yas = "15"
gelecek_yas = yas + 1

# Kod 2
sayi = int("12.5")

# Kod 3
mesaj = "Toplam: " + 250

# Kod 4
sonuc = "5" * "3"
```

Kod No	Fırlatılan İstisna	Hatanın Nedeni
Kod 1		
Kod 2		
Kod 3		
Kod 4		

3. Adım: Üçüncü öğrenme biriminden hatırlayacağınız tür dönüşümü fonksiyonlarını (int(), float(), str()) kullanarak her hatayı doğru hâle getiriniz. Örneğin Kod 1'de yas değişkeni int() ile sayıya çevrilmelidir.

4. Adım: Düzelttiğiniz kodları çalıştırarak doğru sonuç ürettiklerini doğrulayınız.

5. Adım: Dört hatayı, fırlattıkları istisna türüne göre sınıflandırınız. Hangileri TypeError, hangileri ValueError fırlattı? Bir tabloda gruplandırınız.

6. Adım: Çalışmalarınızı README dosyasıyla birlikte dijital deponuza yükleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.



Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



SIRA SİZDE

Aşağıdaki kod kullanıcıdan iki sayı alıp toplamak ister ancak çalıştırıldığında bir hata fırlatır.

```
sayi1 = input("Birinci sayı: ")
sayi2 = input("İkinci sayı: ")
toplam = sayi1 + sayi2
print("Toplam:", toplam)
```

Aşağıdaki soruları cevaplayınız.

1. Kullanıcı 5 ve 3 girdiğinde program ne çıktı verir, neden?
2. Kodu, sayıların gerçekten toplanacağı şekilde tür dönüşümü kullanarak düzeltiniz.

6.2. HATA AYIKLAMA BLOKLARI

Hataların bazıları istisna fırlatmaz. Program sorunsuz çalışır fakat yanlış sonuç üretir. Bu tür mantıksal hataların belirlenebilmesi için programın çalışma sürecinin adım adım izlenmesi gerekir.

Hata ayıklama (debugging), bir programdaki hataları bulma ve düzeltme sürecidir. "Bug" kelimesi İngilizcede "böcek" anlamına gelir. Bilgisayar tarihinde gerçek bir böceğin bir makinede arızaya yol açmasından sonra hatalara "bug" hata bulma sürecine de "debugging" denmeye başlanmıştır.

Özellikle **mantıksal hatalar**, program çökmediği için fark edilmesi en zor hatalardır. Bir program yanlış sonuç üretiyorsa ama hata mesajı vermiyorsa sorunun tam olarak nerede olduğunu anlamak gerekir. İşte bu noktada **hata ayıklama blokları** devreye girer.

Hata ayıklama bloğu, programın çalışması sırasında değişkenlerin o anki değerlerini görmek amacıyla koda eklenen geçici kod parçasıdır. En yaygın ve en basit hata ayıklama bloğu, değişkenlerin değerini print() fonksiyonu ile ekrana yazdıran satırlardır. Bu satırlar sayesinde programın hangi adımda hangi değeri tuttuğu adım adım izlenebilir.

BİLGİ KUTUSU

Hata ayıklama bloklarını **[DEBUG]** gibi bir etiketle işaretlemek iyi bir alışkanlıktır. Bu satırlar daha sonra kolayca bulunup silinebilir veya yorum satırı hâline getirilebilir. Etiket, hata ayıklama satırlarını programın asıl çıktılarından ayırt etmeyi de kolaylaştırır.

Hata ayıklama bloğuyla yalnızca değişken değerleri değil programın **hangi satıra ulaştığı** da izlenebilir. Bir fonksiyonun çağrılıp çağrılmadığını veya bir koşulun çalışıp çalışmadığını kontrol etmek için de DEBUG satırları kullanılır.

15. ÖRNEK UYGULAMA

Amaç: Programın hangi koşula girdiğini hata ayıklama bloğuyla izlemek

```
1  def harf_notu(puan):
2      if puan >= 85:
3          print("[DEBUG] 1. koşula girildi (puan >= 85)")
4          return "AA"
5      elif puan >= 70:
6          print("[DEBUG] 2. koşula girildi (puan >= 70)")
7          return "BA"
8      else:
9          print("[DEBUG] son koşula girildi (puan < 70)")
10         return "FF"
11
12 print("Harf notu:", harf_notu(76))
```

EKRAN ÇIKTISI
[DEBUG] 2. koşula girildi (puan >= 70)
Harf notu: BA

16. ÖRNEK UYGULAMA

Amaç: Bir programa hata ayıklama bloğu ekleyerek değişken değerlerini izlemek

```
1  def ortalama_bul(notlar):
2      toplam = 0
3      for n in notlar:
4          toplam += n
5          print(f"[DEBUG] eklenen={n}, güncel toplam={toplam}")
6      print(f"[DEBUG] eleman sayısı={len(notlar)}")
7      return toplam / len(notlar)
8
9  print("Ortalama:", ortalama_bul([70, 80, 90]))
```

EKRAN ÇIKTISI
[DEBUG] eklenen=70, güncel toplam=70
[DEBUG] eklenen=80, güncel toplam=150
[DEBUG] eklenen=90, güncel toplam=240
[DEBUG] eleman sayısı=3
Ortalama: 80.0

17. ÖRNEK UYGULAMA

Amaç: Hata ayıklama bloğu ile mantıksal hatayı tespit etmek

```
1  def en_buyuk_bul(sayilar):
2      en_buyuk = 0
3      for s in sayilar:
4          print(f"[DEBUG] incelenen={s}, mevcut en_buyuk={en_buyuk}")
5          if s > en_buyuk:
6              en_buyuk = s
7      return en_buyuk
8
9  print("En büyük:", en_buyuk_bul([-5, -12, -3, -8]))
```

EKRAN ÇIKTISI
[DEBUG] incelenen=-5, mevcut en_buyuk=0
[DEBUG] incelenen=-12, mevcut en_buyuk=0
[DEBUG] incelenen=-3, mevcut en_buyuk=0
[DEBUG] incelenen=-8, mevcut en_buyuk=0
En büyük: 0

BİLGİ KUTUSU

Hata ayıklama blokları geçici kod parçalarıdır. Program tamamlanarak hata düzeltildiğinde bu satırlar koddan silinmeli veya yorum satırı hâline getirilmelidir. Aksi hâlde gereksiz çıktılar, programın gerçek sonuçlarını gölgeleyebilir ve kullanıcının kafasını karıştırabilir.

Hata ayıklama bloklarının yanı sıra programcılar hataları bulmak için **yorum satırı** yöntemini de kullanır. Şüphelenilen kod satırının başına # işareti konularak o satır geçici olarak devre dışı bırakılır. Hata ortadan kalkarsa sorunun o satırda olduğu anlaşılır. Bu yöntem, hata ayıklama bloklarıyla kullanıldığında oldukça etkilidir.

Hata Ayıklamada İzlenmesi Gereken İpuçları

- Hata küçük parçalara bölünerek aranır. Tüm programı bir anda incelemek yerine şüpheli bölümlere odaklanılır.
- Değişkenlerin değerleri kritik noktalarda ekrana yazdırlır.
- Programı bilinen sonuçlu basit verilerle test edilir. Örneğin faktöriyel için 4! = 24 olduğu bildiğinden bu değerle denenir.
- Hatayı bulunduktan sonra benzer hataların başka yerlerde de olup olmadığı kontrol edilir.
- Hata ayıklama tamamlandığında tüm geçici DEBUG satırlarını mutlaka temizleyiniz.

BİLGİ KUTUSU

Profesyonel yazılım geliştirme ortamlarında (IDE) hata ayıklama için özel araçlar bulunur. Bu araçlar, programı satır satır çalıştırma ve her adımda tüm değişkenlerin değerini görme imkânı sunar. Ancak print() ile hata ayıklama, hiçbir özel araç gerektirmediği için temel ve evrensel yöntemdir.

18. ÖRNEK UYGULAMA

Amaç: Bir döngüdeki sayacın yanlış ilerlemesini hata ayıklama bloğuyla bulmak

```
1  def pozitif_say(sayilar):
2      sayac = 0
3      for s in sayilar:
4          print(f"[DEBUG] incelenen={s}, sayac={sayac}")
5          if s > 0:
6              sayac = 1
7      return sayac
8
9  print("Pozitif sayı adedi:", pozitif_say([4, -2, 7, 9, -1]))
```

EKRAN ÇIKTISI
[DEBUG] incelenen=4, sayac=0
[DEBUG] incelenen=-2, sayac=1
[DEBUG] incelenen=7, sayac=1
[DEBUG] incelenen=9, sayac=1
[DEBUG] incelenen=-1, sayac=1
Pozitif sayı adedi: 1

SIRA SİZDE

Aşağıdaki kod bir listedeki çift sayıların toplamını bulmaya çalışır ama yanlış sonuç verir.

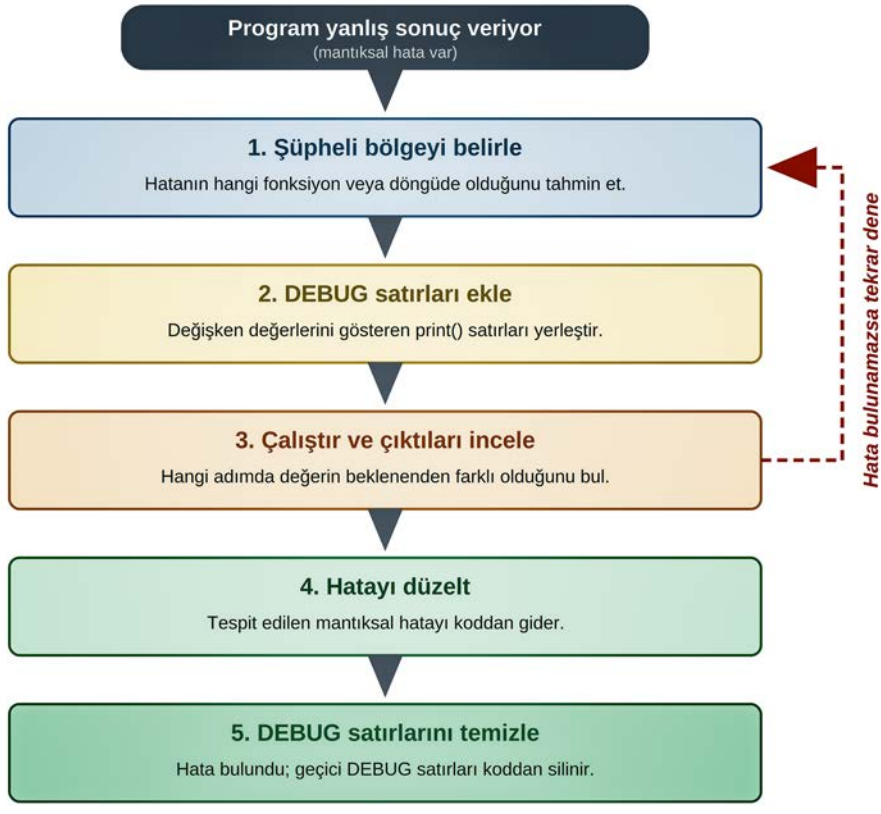
```
def cift_toplam(sayilar):
    toplam = 0
    for s in sayilar:
        if s % 2 == 1:
            toplam += s
    return toplam
print(cift_toplam([1, 2, 3, 4, 5, 6]))
```

Aşağıdaki soruları cevaplayınız.

1. Fonksiyonun içine [DEBUG] etiketli hata ayıklama blokları ekleyerek hangi sayıların toplandığını izleyiniz. Çıktıları inceleyerek mantıksal hatayı bulunuz (İpucu: % 2 == 1 koşulu hangi sayıları seçer?).
2. Hatayı düzeltip doğru sonucu aldığınızı doğrulayınız.

Hata Ayıklama Bloklarının Kullanımı

Hata ayıklama blokları rastgele eklenmez. Belirli bir yöntem izlenerek kullanılır. Bu yöntem hatayı adım adım daraltarak bulmayı sağlar. Hata ayıklama süreci beş aşamadan oluşur, bu aşamalar Görsel 6.5'te gösterilmiştir.



Görsel 6.5: Hata ayıklama sürecinin aşamaları

Önce hatanın olabileceği **şüpheli bölge** belirlenir ve bu bölgeye **DEBUG satırları** eklenir. Program çalıştırılır, çıktılar incelenir, hatanın oluştuğu nokta bulunur hata düzeltilir ve son olarak **DEBUG satırları temizlenir**. Hata ilk denemede bulunamazsa farklı bir bölgeye DEBUG satırları eklenerek süreç tekrarlanır.

19. ÖRNEK UYGULAMA

Amaç: Hata ayıklama sürecini adım adım uygulayarak bir hatayı bulmak

```
1   def ortalama(n1, n2, n3):
2       toplam = n1 + n2 + n3
3       sonuc = toplam / 2
4       return sonuc
5
6   print("Ortalama:", ortalama(60, 70, 80))
```

EKRAN ÇIKTISI
Ortalama: 105.0

Sonuç yanlıştır. 105 yerine 70 beklenmektedir. Şüpheli bölgeye DEBUG satırı eklenir.

```python
def ortalama(n1, n2, n3):
    toplam = n1 + n2 + n3
    print(f"[DEBUG] toplam={toplam}")
    sonuc = toplam / 2
    print(f"[DEBUG] sonuc={sonuc}")
    return sonuc

print("Ortalama:", ortalama(60, 70, 80))
```

EKRAN ÇIKTISI
[DEBUG] toplam=210
[DEBUG] sonuc=105.0
Ortalama: 105.0

DEBUG çıktılarıyla toplamın doğru hesaplandığı (210) ama toplamın **2'ye bölündüğü gösterilir**. Oysa üç notun ortalaması için **3'e** bölünmesi gerektiği belirtilir. Hatanın bulunduğu görülür: **toplam / 2** yerine **toplam / 3** yazılmalıdır. Düzeltme yapıldıktan sonra DEBUG satırları silinir.

20. ÖRNEK UYGULAMA

Amaç: İki farklı hatayı hata ayıklama bloğuyla aşamalı olarak bulmak

Aşağıdaki kodla bir listedeki sayıların ortalamasının iki katı hesaplanmaya çalışılır ve [70, 80, 90] listesi için beklenen sonuç 160'tır.

```python
def ortalama_iki_kat(sayilar):
    toplam = 0
    for s in sayilar:
        toplam += s
    ortalama = toplam / len(sayilar)
    sonuc = ortalama + 2
    return sonuc

print(ortalama_iki_kat([70, 80, 90]))
```

EKRAN ÇIKTISI
82.0

Sonuç yanlış olduğu (82, oysa 160 beklenmektedir) belirtilir. Hata ayıklama bloğu eklenir.

```python
def ortalama_iki_kat(sayilar):
    toplam = 0
    for s in sayilar:
        toplam += s
    print(f"[DEBUG] toplam={toplam}")
    ortalama = toplam / len(sayilar)
    print(f"[DEBUG] ortalama={ortalama}")
    sonuc = ortalama + 2
    print(f"[DEBUG] sonuc={sonuc}")
    return sonuc

print(ortalama_iki_kat([70, 80, 90]))
```

EKRAN ÇIKTISI
[DEBUG] toplam=240
[DEBUG] ortalama=80.0
[DEBUG] sonuc=82.0

Düzeltmeden sonra sonuç 160 olur ve DEBUG satırları temizlenir.

SIRA SİZDE

1. Aşağıdaki kod bir sayının faktöriyelini hesaplamaya çalışır ama yanlış sonuç verir. 4! = 24 beklenirken farklı bir sonuç çıkar.

```
def faktoriyel(n):
    sonuc = 0
    for i in range(1, n + 1):
        sonuc = sonuc * i
    return sonuc
print(faktoriyel(4))
```

Programı yazarak aşağıda verilen hata ayıklama sürecini uygulayınız.

1. Adım: Şüpheli bölgeyi belirleyiniz.

2. Adım: sonuc değişkenini izleyen [DEBUG] satırları ekleyiniz.

3. Adım: Çıktıları inceleyiniz.

4. Adım: Mantıksal hatayı bulup düzeltiniz.

5. Adım: DEBUG satırlarını temizleyiniz. Hatanın **sonuc** değişkeninin başlangıç değeriyle olan ilişkisini yazınız.

2. Aşağıdaki kod 1'den n'e kadar olan sayıların toplamını bulmaya çalışır. n=5 için 15 beklenirken yanlış sonuç verir.

```
def toplam_bul(n):
    toplam = 0
    for i in range(1, n):
        toplam += i
    return toplam
print(toplam_bul(5))
```

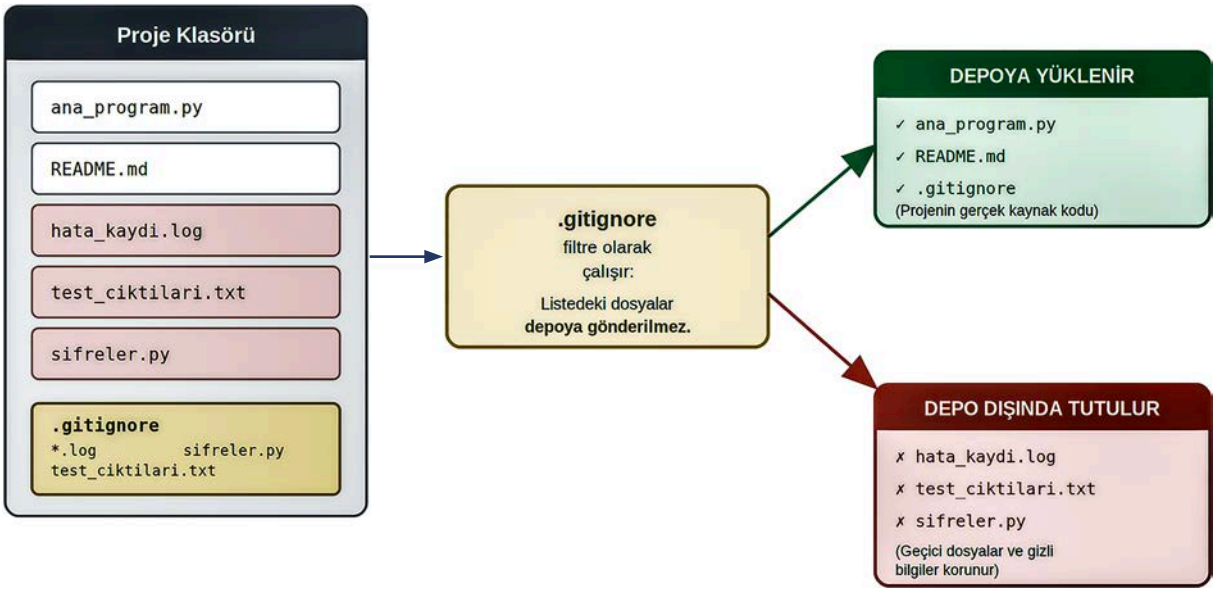
Aşağıdaki işlemleri yapınız.

a) Programa [DEBUG] satırları ekleyerek döngünün hangi sayıları topladığını izleyiniz.

b) Mantıksal hatayı bulunuz. Hatayı düzelttikten sonra DEBUG satırlarını temizleyiniz.

Geçici Dosyalar ve .gitignore Kullanımı

Hata ayıklama sırasında bazen geçici dosyalar oluşturulur (örneğin DEBUG çıktılarının kaydedildiği günlük dosyaları). Bu geçici dosyaların projenin dijital deposuna karışmaması için **.gitignore** dosyası kullanılır. .gitignore, dijital depoya **yüklenmemesi gereken** dosyaların listelendiği özel bir metin dosyasıdır. Bu dosya sayesinde veri bütünlüğü korunur ve sürüm kontrolü düzenli tutulur. .gitignore dosyasının çalışma mantığı Görsel 6.6'da gösterilmiştir.



Görsel 6.6: .gitignore dosyasının çalışma mantığı.

Bir .gitignore dosyası örneği aşağıda verilmiştir.

```
# Hata ayıklama günlük dosyaları
*.log
debug_ciktilari.txt

# Test sırasında oluşan geçici dosyalar
geçici /
```

Buradaki ***.log** ifadesi "uzantısı .log olan tüm dosyalar" anlamına gelir. Bu dosyada adı geçen ögeler dijital depoya yüklenmez böylece depo yalnızca projenin gerçek kaynak kodunu içerir.

BİLGİ KUTUSU

gitignore dosyasının adı bir noktayla başlar ve uzantısı yoktur. Bu dosya, projenin ana klasörüne yerleştirilir. İçine her satırda bir dosya adı, klasör adı veya desen yazılır. Hata ayıklama sırasında oluşturduğunuz geçici çıktı dosyalarını .gitignore'a eklemek iyi bir alışkanlıktır.

6. UYGULAMA

Mantıksal Hataları [DEBUG] Çıktılarıyla Gideriyorum

Süre: 40 dakika

Amaç: Hata ayıklama bloklarını kullanmak

Yönerge: Aşağıdaki adımları takip ederek hata ayıklama bloklarını kullanınız.

1. Adım: Editörünüzde **hata_ayiklama.py** adında yeni bir dosya oluşturunuz.

2. Adım: Aşağıdaki mantıksal hatalı kodu dosyaya kopyalayınız. Bu kod bir sayının faktöriyelini hesaplamaya çalışır ama yanlış sonuç üretir.

```
def faktoriyel(n):
    sonuc = 0
    for i in range(1, n + 1):
        sonuc = sonuc * i
    return sonuc

print("5! =", faktoriyel(5))   # Beklenen sonuç: 120
```

3. Adım: Fonksiyonun içine **[DEBUG]** etiketli hata ayıklama blokları ekleyerek döngünün her adımında sonuc ve i değişkenlerinin hangi değeri aldığını izleyiniz. .

4. Adım: Programı çalıştırıp DEBUG çıktılarını inceleyiniz. Hangi adımda beklenmedik bir değer oluştuğunu tespit ediniz.

5. Adım: Mantıksal hatayı bulunuz. Bir sayı sürekli 0 ile çarpılırsa sonuç ne olur? Sonuç değişkeninin başlangıç değeri ne olmalıydı?

6. Adım: Hatayı düzeltiniz ve programı yeniden çalıştırarak doğru sonucu (120) aldığınızı doğrulayınız.

7. Adım: Hata bulunup düzeltildikten sonra tüm [DEBUG] satırlarını programdan temizleyiniz.

8. Adım: Proje klasörünüzde bir .gitignore dosyası oluşturunuz. Hata ayıklama sırasında oluşabilecek geçici dosyaları (örneğin *.log ve debug_ciktilari.txt) bu dosyaya ekleyiniz.

9. Adım: Düzeltilmiş programınızı ve .gitignore dosyanızı README dosyasıyla birlikte dijital deponuza yükleyiniz. README dosyasında bulduğunuz mantıksal hatayı ve nasıl düzelttiğinizi açıklayınız.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.



Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



BİLGİ KUTUSU

Güvenilir Yazılım Geliştirme Kültürü

Yazılım geliştirme süreci sadece kodun o an çalışmasını sağlamaktan ibaret değildir. Profesyonel bir yazılımcı, kodunun beklenmedik durumlarda ve hatalı kullanımlarda nasıl davranacağını önceden planlar. Kapsamlı bir projeye başlamadan önce yazılım sektöründe benimsenen bazı altın kuralları hatırlamak oldukça önemlidir.

- **Hatalar Düşman Değil, Yol Göstericidir:** Hatalar programlamanın doğal bir parçasıdır. Karşılaştığınız her hata mesajı, programı daha dayanıklı ve güvenli hâle getirmek için bir fırsattır.
- **Kullanıcı Deneyimini Önemseyiniz:** Son kullanıcılar yazılım dillerini bilmek zorunda değildir. Onların karşısına teknik hata terimleri (ValueError, IndexError vb.) çıkarmak yerine neyin yanlış yaptıklarını ve durumun nasıl düzeltilebileceğini açıklayan net, yönlendirici ve anlaşılır mesajlar sunulmalıdır.
- **Hataları Halının Altına Süpürmeyiniz:** İstisnaları yakalayıp except bloğunun içini boş bırakmak (hatayı sessizce geçiştirmek) en tehlikeli alışkanlıklardan biridir. Yakalanan bir hata programı çökertmese bile mutlaka kayıt altına alınmalı (loglanmalı) veya kullanıcıya bildirilmelidir.
- **İz Bırakmadan Çalışınız:** Hata ayıklarken [DEBUG] mesajları kullanmak mükemmel bir sorun çözme yöntemidir ancak projenizi bitirip teslim etmeden önce bu geçici test satırlarını temizlemek ve .gitignore ile gereksiz günlük dosyalarının depoya gitmesini engellemek profesyonelliğin temel bir göstergesidir.

6.1. PROJE	
Konu	**Güvenli Kullanıcı Kayıt Sistemi Geliştiriyorum**
Kapsanan Konular	Hata yakalama, istisna yönetimi
Süre	40+40+40 dakika
Amaç	Hata yakalama işlemlerini kapsayıcı bir projede kullanmak
Çalışma Biçimi	Bireysel

Proje Senaryosu

Kütüphane otomasyon sistemi için kullanıcı kayıt programı geliştiriniz. Program; kullanıcıdan ad, yaş, e-posta, telefon ve üyelik türü bilgilerini alır. Girilen verilerin doğruluğunu try-except bloklarıyla kontrol eder; boş, eksik ya da yanlış formattaki verileri kullanıcıya bildirir ve yeniden ister.

Öğrendiğiniz istisna işlemleri, try-except-else-finally, raise, hata ayıklama blokları ve .gitignore konularının tamamını bu projede bir arada kullanacaksınız.

Yönerge: Aşağıdaki adımları takip ederek projenizi geliştiriniz. Proje genelinde aşağıdaki beceri, eğilim ve programlar arası bileşenler ilişkilendirilmiştir:

1. Adım: Projenizin algoritmasını yazınız. Hangi verinin (yaş, e-posta, telefon, üyelik türü) hangi hatayı verebileceğini ve bu hata için hangi istisnayı kullanacağınızı önceden bir tablo hâlinde belirleyiniz.

2. Adım: Editörünüzde **kayit.py** adında bir dosya oluşturunuz.

3. Adım (Yaş Kontrolü): Kullanıcıdan yaş bilgisini alan bir bölüm yazınız. Yaş sayı değilse ValueError yakalayınız; 0-120 aralığı dışındaysa **raise** ile uygun bir istisna fırlatınız.

4. Adım (E-posta Kontrolü): Kullanıcıdan e-posta bilgisini alınız. e-posta @ işareti içermiyorsa raise ile bir istisna fırlatıp kullanıcıya "Geçersiz e-posta biçimi" mesajı veriniz.

5. Adım (Telefon Kontrolü): Kullanıcıdan telefon numarasını alınız. Telefon **boş** bırakılırsa veya rakam dışında karakter içeriyorsa uygun istisnayı yakalayınız.

6. Adım (Üyelik Türü Kontrolü): Üyelik türlerini bir **sözlük** içinde tutunuz (örneğin **{"1": "Standart", "2": "Öğrenci", "3": "Premium"}**). Kullanıcı listede olmayan bir seçim yaparsa **KeyError** yakalayınız. **7. Adım (Ana Program Akışı):** Tüm kayıt işlemini bir **while True döngüsü** içine alınız. Her bilgi alımını **try-except** blokları ile yöneterek kullanıcının hatalı giriş yaptığında o adımı tekrar denemesini sağlayınız.

8. Adım (finally ile Özet): Her kayıt denemesinin sonunda, hata olsa da olmasa da çalışan bir **finally** bloğu yazınız. Bu blok "Kayıt işlemi tamamlandı." bilgisini ve başarılıysa kaydedilen kullanıcı özetini göstersin.

9. Adım (.gitignore Oluşturma): Proje klasörünüzde, **.gitignore** dosyası oluşturunuz. Test sırasında oluşturduğunuz geçici dosyaları (örneğin **test_kullanicilar.txt, *.log**) bu dosyaya ekleyerek depoya yüklenmelerini engelleyiniz.

10. Adım (Test): Programınızı; boş giriş, sayı yerine harf, yaş aralığı dışı değer, yanlış e-posta biçimi, geçersiz üyelik türü seçimi olmak üzere en az **beş farklı senaryo** ile test ediniz.

11. Adım (Dijital Depo ve Issue Açma): Tüm dosyaları (kayit.py, .gitignore, README.md) dijital deponuza yükleyiniz. README.md dosyasında projenin işlevini ve hangi istisnaların ne için kullanıldığını açıklayınız. Ardından depo platformunuzun **"Issues" (Sorunlar)** bölümüne giderek projenize en az **iki Issue** oluşturunuz. Şu şekilde oluşturabilirsiniz:

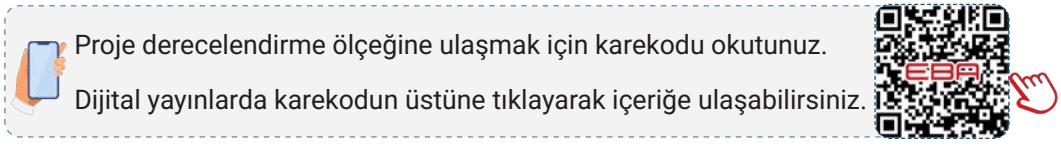
- Issue #1 (enhancement): "Telefon numarası için 11 hane uzunluk kontrolü eklenmeli."
- Issue #2 (enhancement): "Aynı e-posta ile ikinci kez kayıt engellenmeli."

Her Issue için **başlık, açıklama** ve uygun bir **etiket** (enhancement, bug, documentation vb.) belirleyiniz.

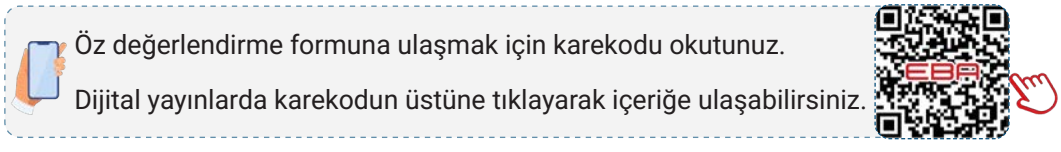
12. Adım (Issue İçin Çözüm Commit'i): Açtığınız Issue'lardan en az birine yönelik kodda gerekli iyileştirmeyi yapınız. Değişikliğinizi commit ederken **commit mesajında Issue numarasına referans veriniz** (Örneğin "Fixes #1: telefon numarası uzunluk kontrolü eklendi."). Bu sayede hangi commit'in hangi sorunu çözdüğü depo üzerinden izlenebilir hâle gelir.

Değerlendirme

a) Yapacağınız proje çalışması öğretmeniniz tarafından **Proje Derecelendirme Ölçeği** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak proje değerlendirme ölçeğine ulaşabilirsiniz.



b) **Öz Değerlendirme Formu** proje geliştirme sürecinin sonunda kendinizi değerlendirmeniz amacıyla hazırlanmıştır. Cevaplarınızı tabloda ilgili kutucuğa yazınız. Verilen karekodu kullanarak öz değerlendirme formuna ulaşabilirsiniz.



ÖLÇME VE DEĞERLENDİRME

A) Aşağıdaki cümlelerde boş bırakılan yerleri doğru kelimelerle doldurunuz.

1. Program çalışırken ortaya çıkan ve normal akışı kesen beklenmedik olaya ..... denir.

2. Bir istisna oluştuğunda Python'un ekrana yazdırdığı hata raporuna ..... denir.

3. Riskli kod ..... bloğunun içine yazılır.

4. İstisna oluştuğunda çalışacak kod ..... bloğunun içine yazılır.

5. Bir sayının sıfıra bölünmesi ..... istisnasını fırlatır.

6. Var olmayan bir sözlük anahtarına erişildiğinde ..... istisnası oluşur.

7. İstisna olsa da olmasa da her durumda çalışan blok ..... bloğudur.

8. Programdaki hataları bulma ve düzeltme sürecine ..... denir.

9. Değişken değerlerini izlemek için koda eklenen kod parçasına ..... bloğu denir.

10. Dijital depoya yüklenmemesi gereken dosyalar ..... dosyasında listelenir.

B) Sol sütundaki açıklamayı, sağ sütundaki doğru istisna türüne ait harfle eşleştiriniz.

	Açıklama
( )	1. Var olmayan bir sözlük anahtarına erişme
( )	2. Bir sayıyı sıfıra bölme
( )	3. Listede olmayan bir indekse erişme
( )	4. int("abc") gibi uygunsuz değer dönüşümü
( )	5. Metin ile sayıyı toplama

İstisna Türü
a) ZeroDivisionError
b) ValueError
c) KeyError
ç) IndexError
d) TypeError
e) FileNotFoundError

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1.

```
try:
    print(10/0)
except ZeroDivisionError:
    print("Hata!")
```

Yukarıdaki kodun çıktısı aşağıdakilerden hangisidir?

A) 0
B) 10
C) Hata!
D) Traceback
E) None

2. int(input(“Sayı: ”)) ifadesi, kullanıcı “abc” yazarsa aşağıdaki hangi istisnayı fırlatır?

A) IndexError
B) KeyError
C) NameError
D) ValueError
E) TypeError

3. stok = {“ekmek”: 50} sözlüğünde stok[“sut”] ifadesi aşağıdaki hangi istisnayı fırlatır?

A) IndexError
B) KeyError
C) NameError
D) ValueError
E) TypeError

4.

```
try:
    sonuc = 10/2
except ZeroDivisionError:
    print(“Hata”)
finally:
    print(“BİTTİ”)
```

Yukarıdaki kodda “BİTTİ” mesajı kaç defa yazılır?

A) 0
B) 1
C) 2
D) 3
E) Hiç bir çıktı olmaz.

5. finally bloğu ile ilgili aşağıdakilerden hangisi doğrudur?

A) Sadece istisna oluşunca çalışır.
B) Sadece istisna oluşmayınca çalışır.
C) Hiçbir zaman çalışmaz.
D) Sadece except bloğundan önce çalışır.
E) İstisna olsa da olmasa da her durumda çalışır.

6.

```
try:
   yas = int(input("Yaş: "))
except ValueError:
   print("Geçersiz!")
else:
   print("Kabul edildi.")
```

Yukarıdaki kodun davranışı ile ilgili aşağıdaki anahtar sözcüklerden hangisi kullanılır?

A) Kullanıcı 20 girerse "Geçersiz!" yazılır.
B) Kullanıcı 20 girerse "Kabul edildi." yazılır.
C) Kullanıcı "abc" girerse "Kabul edildi." yazılır.
D) Her durumda program çöker.
E) else bloğu hiç çalışmaz.

7. Bir istisnayı bilinçli olarak fırlatmak için hangi anahtar sözcük kullanılır?

A) catch
B) except
C) finally
D) raise
E) try

8. Hata ayıklama bloğu (DEBUG satırı) ile ilgili aşağıdakilerden hangisi doğrudur?

A) Değişken değerlerini izlemek için eklenir, hata bulununca temizlenir.
B) Programın kalıcı bir parçasıdır, asla silinmez.
C) Sadece söz dizimi hatalarını bulur.
D) Programı hızlandırır.
E) İstisna fırlatır.

9. Bir program çalışıyor, hata mesajı vermiyor ama yanlış sonuç üretiyor. Bu hata türü aşağıdakilerden hangisidir?

A) Söz dizimi hatası
B) Çalışma zamanı hatası
C) İstisna
D) Mantıksal hata
E) Traceback

10. Hata ayıklarken oluşan geçici dosyaların dijital depoya yüklenmesini önlemek için hangi işlem yapılır?

A) Dosyalar silinir.
B) try-except kullanılır.
C) raise kullanılır.
D) Hata ayıklama bloğu eklenir.
E) .gitignore dosyasına eklenir.

Ç) 1-3. soruları aşağıdaki metne göre cevaplayınız.

Derya kullanıcıdan alınan verileri işleyen ve liste fonksiyonları barındıran bir Python programı yazar. Kodun sorunsuz çalışması ve mantıksal hatalardan arınması için uyguladığı adımlar ve yazdığı kod parçaları aşağıda verilmiştir.

1. Adım

```
try:
    yas = int(input("Yaşınızı giriniz: "))
except:
    print("Bir hata oluştu!")
```

Kullanıcı "yirmi" gibi metinsel bir ifade girdiğinde programın çökmesinin engellendiğini ve en doğru hata yönetimi yapısını kullandığını düşünür.

2. Adım: Daha önce oluşturduğu şehirler listesinden kullanıcının seçtiği indeksteki şehri ekrana yazdırmak için bir fonksiyon tasarlar. Listede olmayan bir indeks girildiğinde programın çöktüğünü fark eder. Hatanın hangi satırdan ve değişkenden kaynaklandığını adım adım izlemek amacıyla döngünün içine kalıcı olarak kalmasını planladığı

```
print("[DEBUG] Mevcut indeks:", indeks)
```

satırını ekler.

1. Geliştirme ve test etme aşamasında yaptığı hatalı işlemler hangileridir? Bu adımlardaki söz dizimi, çalışma zamanı ve mantıksal hataları açıklayınız.

2. Bu hataların düzeltilmesi, istisnaların doğru yönetilmesi ve hata ayıklama sürecinin sağlıklı tamamlanması için her adımda ne tür değişiklikler yapılmalıdır?

3. Geliştirdiğiniz yazılımlarda mantıksal hataların programı çökertmediğini göz önünde bulundurunuz. Projelerinizin teslim aşamasına gelmeden önce kod temizliği ve güvenilir test senaryoları oluşturmak için nasıl bir strateji izlemelisiniz?

D) Bu öğrenme biriminde edindiklerinizi değerlendirmek için aşağıdaki çıkış kartını doldurunuz.

Üç Yaz	**Hata Yakalama İşlemleri öğrenme biriminde öğrendiğiniz veya önemli bulduğunuz üç bilgiyi yazınız.**
1	
2	
3	
İki Sor	**İstisna işlemleri veya hata ayıklama blokları ile ilgili merak ettiğiniz iki soruyu yazınız.**
1	
2	
Bir Paylaş	**Bu öğrenme biriminde geliştirdiğiniz uygulamalardan birini sınıf arkadaşlarınızla paylaşmak isteseniz hangisini seçerdiniz? Neden?**



KONULAR

7.1. NESNE VE SINIFLAR

7.2. MİRAS ALMA İŞLEMLERİ

7.3. ÇOK BİÇİMLİ FONKSİYONLAR

7.4. SOYUT SINIFLAR

ANAHTAR KAVRAMLAR

- çok biçimlilik
- kapsülleme
- miras alma
- nesne
- nesne yönelimli programlama
- nitelik
- sınıf
- soyut sınıf



7. ÖĞRENME BİRİMİ

NESNE YÖNELİMLİ PROGRAMLAMA (OOP)

BEKLENTİLER

Bu öğrenme biriminde sizden

- nesneleri ve sınıfları kullanarak program yazmanız
- program içerisinde miras alma işlemlerini gerçekleştirmeniz
- çok biçimli fonksiyonları yazmanız
- soyut sınıfları kodlamanız

beklenmektedir.

HAZIRLIK ÇALIŞMALARI

1. Aşağıda bir yazılım geliştiricinin günlüğünden bir bölüm verilmiştir.

Tasarım yapmayı seven bir arkadaşım bir mobil oyun geliştirdi. Oyunda onlarca farklı karakter var. Her karakterin adını, şeklini, enerjisini ve hızını tanımlayarak karakter özellikleri için ayrı değişkenler oluşturdu. Karakterler için yürüme, zıplama gibi hareket kodlarını her karakterde alt alta tekrar tekrar yazdı. Oyuna yeni karakter eklendiğinde yüzlerce satır kodun içinde kayboldu. Karakterlere koşma kabiliyeti eklemeye karar verdi ama bu kabiliyeti tek tek uygulaması gerekti. Elinde iyi bir oyun fikri olmasına karşın programı istediği gibi tasarlayamadı.

Verilen metne göre aşağıdaki soruları cevaplayınız.

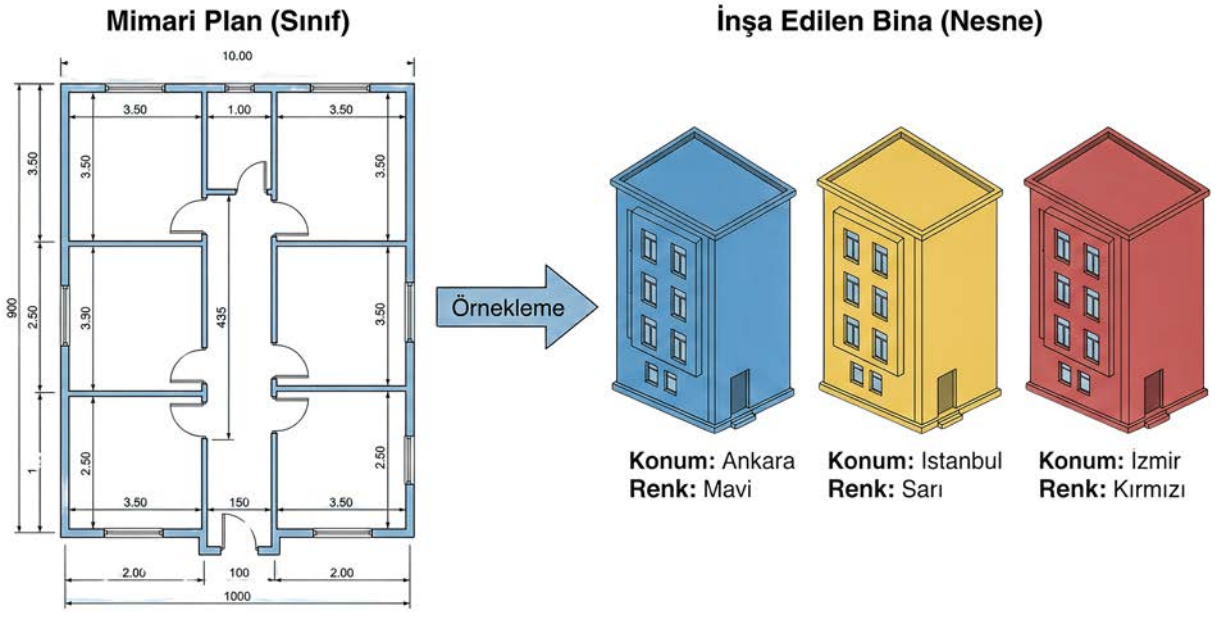
a) Yukarıdaki metinden hareketle yazılım geliştirici, oyuna yeni özellikler eklemek veya var olanları değiştirmek istediğinde hangi sorunlarla karşılaştı?

b) Daha önce öğrendiğiniz modüler programlama ilkelerini kullanarak kod tekrarını önleyebilir misiniz? Bu yazılım geliştiriciye nasıl bir çözüm önerirsiniz?

BAŞLARKEN

Bir mimari proje çiziminde odaların boyutları, kapı ve pencerelerin konumları ile kullanılacak yapı malzemeleri ayrıntılı olarak belirlenir. Mimarın hazırladığı bu plan, bir binanın nasıl inşa edileceğini gösteren bir **şablon**dur. Aynı mimari plan kullanılarak farklı yerlerde binalar inşa edilebilir. Bu binalar aynı plana dayanmakla birlikte bulundukları yer ve renkleri bakımından birbirinden farklı olabilir.

Programlama dünyasında da benzer bir durum vardır. **Sınıf (class)** bir tür plan veya şablon, **nesne (object)** ise bu mimari plana göre oluşturulan somut yapılardır (Görsel 7.1).


Mimari Plan (Sınıf)
İnşa Edilen Bina (Nesne)
10.00
3.50
1.00
3.50
3.50
3.50

da karmaşık problemleri çözmek ve gerçek hayatı dijital ortama aktarmak için benzer bir mantık kullanılır. Örneğin bilgisayar oyunlarındaki yüzlerce farklı karakter her defasında tekrar kodlanmaz. Bunun yerine ana şablonlar üzerinden türetilen somut varlıklar olarak modellenir.

7.1. NESNE VE SINIFLAR

Nesne yönelimli programlama [Object Oriented Programming (OOP)], programlama ortamında gerçek dünyayı modellemek için geliştirilen bir yaklaşımdır. Bu yaklaşımda yazılım tasarlanırken **özellikler** (attributes) ve **metotlar** (methods) barındıran **nesneler** kullanılır.

Bir nesnenin nasıl oluşturulacağını tanımlayan şablona **sınıf** denir. Sınıf, bir nesnenin hangi özelliklere (niteliklere) ve hangi davranışlara (metotlara) sahip olacağını belirtir. Programda kullanılacak nesne bu şablondan üretilir. Nesne yönelimli programlamada kullanılan temel kavramlar Tablo 7.1'de listelenmiştir.

Tablo 7.1: Nesne Yönelimli Programlama Kavramları

Nesne ve Sınıf Kavramları	Gerçek Dünya Karşılığı	Programlamadaki Karşılığı
Sınıf (Class)	Araba planı / şablonu	class Araba:
Nesne (Object)	Üretilen somut araba	araba1 = Araba(...)
Nitelik (Attribute)	Renk, model, yıl	self.renk = 'Kırmızı'
Metot (Method)	Hareket etme, fren yapma	def sur(self):

Sınıf Tanımlama

Sınıf program içerisinde **class** anahtar sözcüğü ile tanımlanır. Sınıfın adı büyük harfle başlar, içindeki ilk metot olan __init__() **yapıcı metot**tur. Bu sınıftan bir **nesne** oluşturulduğunda otomatik olarak çalışır ve nesnenin niteliklerini başlatır.

1. ÖRNEK UYGULAMA

Amaç: Sınıf ve nesne tanımlamak

```
1   class Araba:
2       def __init__(self, marka, model, yil):
3           self.marka = marka
4           self.model = model
5           self.yil = yil
6       def bilgi_ver(self):
7           print(f"{self.yil} model {self.marka} {self.model}")
8       def calistir(self):
9           #Arabanın motorunu çalıştıran metot.
10          print(f"{self.marka} {self.model} çalıştırıldı..")
11      def fren_yap(self):
12          #Arabanın fren yapmasını sağlayan metot.
13          print(f"{self.marka} {self.model} hız kesiyor ve fren yapıyor!")
14  # Nesne oluşturma
15  araba1 = Araba("Marka1", "Model1", 2026)
16  araba1.bilgi_ver()
17  araba1.calistir()
```

EKRAN ÇIKTISI

```
2026 model Marka1 Model1
Marka1 Model1 çalıştırıldı..
```

BİLGİ KUTUSU

Metodun hangi nesneye ait olduğunu bildiren **self parametresi** her metot tanımında ilk parametre olarak yazılır. Nesne üzerinden metot çağrılırken otomatik olarak iletilen self parametresinin ayrıca belirtilmesine gerek yoktur.

Nesne Niteliklerine Erişim

Sınıf adının ardından gerekli argümanlar verilerek **nesne örneği** (instance) oluşturulur. Oluşturulan nesnenin niteliklerine nokta (.) ile erişilir ve bu nitelikler güncellenebilir.

2. ÖRNEK UYGULAMA

Amaç: Nesnenin niteliklerine erişmek ve niteliklerini güncellemek

```
1   class Araba:
2       def __init__(self, marka, model, yil):
3           self.marka = marka
4           self.model = model
5           self.yil = yil
6   araba1 = Araba('Marka1', 'Model1', 2026)
7   araba2 = Araba('Marka2', 'Model2', 2025)
8   # Niteliklere erişme
9   print(araba1.marka)
10  print(araba2.yil)
11  # Nitelik güncelleme
12  araba1.yil = 2024
13  print(araba1.yil)
```

EKRAN ÇIKTISI
Marka1
2025
2024

1. ETKİNLİK: Nesneleri Modelliyorum

Süre: 40 dakika

Amaç: Fiziksel nesneleri modellemek

Yönerge: Verilen adımları takip ederek uygulamayı tamamlayınız.

1. Adım: Çevrenizdeki veya sınıfınızdaki modellenebilir araçları inceleyiniz. Dijital ortama aktarmak (modellemek) istediğiniz akıllı tahta, sıra veya çanta gibi bir fiziksel nesne seçiniz.

2. Adım: Çevreni modelle diyagramına seçtiğiniz nesnenin adını yazınız ve taslak çizimini yapınız. Sınıfın adını, sahip olacağı nitelikleri ve yapabileceği eylemleri temsil eden metotları belirleyerek yazınız.

3. Adım: Bulduğunuz sonuçları sınıfta arkadaşlarınızla paylaşınız.

4. Adım: Tanımladığınız nesne ve sınıfı bilgisayarda kodlayınız.

5. Adım: Yazdığınız programa göre aşağıdaki soruları cevaplayınız.

Çevreni Modelle	
Seçtiğim Fiziksel Nesne	**Sınıf Adı**
Nesnenin Taslak Çizimi	**Nitelikler / Metotlar**

1. Sınıfınızı tanımlarken isimlendirme kurallarına (örneğin ilk harfin büyük olması) dikkat ederek kodunuzu nasıl başlattınız?

.....

.....

.....

.....

2. Nesnenizin başlangıç özelliklerini (niteliklerini) atamak için **__init__** metodunu ve **self** parametresini, kodunuzda nasıl yapılandırdınız?

.....

.....

.....

.....

3. Seçtiğiniz fiziksel nesnenin **niteliklerini** ve **metotlarını** sınıf içinde nasıl tanımladınız?

.....

.....

.....

.....

4. Tasarladığınız bu genel **sınıf şablonunu** (kalıbını) kullanarak bellekte yer kaplayan **somut bir nesneyi** nasıl ürettiniz?

.....

.....

.....

.....

5. Ürettiğiniz bu somut nesnenin niteliklerine erişmek için hangi yazım kuralını [nokta gösterimi (dot notation)] kullandınız?

.....

.....

.....

.....

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Varsayılan Nitelik Değerleri

Bazı nitelikler, nesne oluşturulurken dışarıdan değer almak yerine otomatik bir başlangıç değeriyle tanımlanabilir. Bu durum, her nesne için ayrı ayrı belirtilmesi gerekmeyen ortak başlangıç değerlerinin kolayca ayarlanmasını sağlar.

3. ÖRNEK UYGULAMA

Amaç: Nesnenin varsayılan değerini değiştirmek

```
1   class Araba:
2       def __init__(self, marka, yil):
3           self.marka = marka
4           # Varsayılan kilometre değeri
5           self.kilometre = 0
6       def kilometre_goster(self):
7           print(f"Araç Kilometresi: {self.kilometre} km")
8       def kilometre_guncelle(self, yeni_km):
9           self.kilometre = yeni_km
10  araba1 = Araba('Marka1', 2026)
11  araba1.kilometre_goster()
12  # Kilometre değeri güncellenebilir.
13  araba1.kilometre_guncelle(15000)
14  araba1.kilometre_goster()
```

EKRAN ÇIKTISI
Araç Kilometresi: 0 km
Araç Kilometresi: 15000 km

1. UYGULAMA

İlk Sınıfımı Kodluyorum

Süre: 40 dakika

Amaç: Nesneleri ve sınıfları kullanan bir program yazmak

Yönerge: Yazılan adımları takip ederek uygulamayı tamamlayınız.

1. Adım: Evcil hayvanları kaydeden bir program yazılmaktadır. Bu programa ait **Hayvan** sınıfını tanımlayınız.

2. Adım: Sınıfa şu nitelikleri ekleyiniz: **ad, tür, yaş** ve **ses (varsayılan değer: 'Bilinmiyor')**.

3. Adım: ses_cikar() adında bir metot yazınız. Bu metot hayvanın çıkardığı sesi ekrana yazdırsın.

4. Adım: Köpek ve Kedi nesneleri oluşturunuz. Özelliklerini tanımlayarak ses_cikar() metodunu çağırınız.

5. Adım: Yazdığınız programı çalıştırarak test ediniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

SIRA SİZDE

Verilen programın çıktısını bilgisayarda kodlamadan tahmin ediniz. Ekran çıktısını yazarak gerekçeleriyle açıklayınız.

```
class Ogrenci:
    def __init__(self, ad, sinif):
        self.ad = ad
        self.sinif = sinif
    def tanitim(self):
        print(f"{self.ad}, {self.sinif}. sınıf öğrencisidir.")
o1 = Ogrenci('Ayşe', 9)
o2 = Ogrenci('Mehmet', 10)
o1.tanitim()
o2.tanitim()
```

EKRAN ÇIKTISI

GEREKÇE

Kapsülleme

Bir sınıftaki verileri koruyucu bir kalkanın içine alarak dışarıdan gelebilecek doğrudan veya hatalı müdahalelere karşı gizleme işlemine **kapsülleme** (encapsulation) denir. Bir nesnenin önemli özellikleri dışarıdan rastgele değiştirilmeye kapatılır ve bu verilere sadece belirlenen güvenlik kuralları çerçevesinde erişilir. Böylece program daha güvenli ve düzenli hâle gelir.

4. ÖRNEK UYGULAMA

Amaç: Kapsülleme ile **veriyi koruyan** bir program yazmak

```
class BankaHesabi:
    def __init__(self):
        self.__bakiye = 0
    def bakiye_goster(self):
        return f"Güncel bakiyeniz: {self.__bakiye} TL"
    def para_yatir(self, miktar):
        if miktar > 0:
            self.__bakiye += miktar
            print(f"{miktar} TL başarıyla yatırıldı.")
        else:
            print("Hata: Yatırılacak miktar 0'dan büyük olmalıdır!")
benim_hesabim = BankaHesabi()
benim_hesabim.para_yatir(150)
print(benim_hesabim.bakiye_goster())
benim_hesabim.para_yatir(-100)
print(benim_hesabim.bakiye_goster())
```

EKRAN ÇIKTISI
150 TL başarıyla yatırıldı.
Güncel bakiyeniz: 150 TL
Hata: Yatırılacak miktar 0'dan büyük olmalıdır!
Güncel bakiyeniz: 150 TL

DİKKAT

Bir değişkenin başına iki alt çizgi eklendiğinde (örneğin __bakiye), bu değişken özel (private) olarak işaretlenir ve dışarıdan doğrudan erişime kapatılır ancak Python'da mutlak bir güvenlik veya kırılamaz bir gizlilik yoktur. Bu işlem değişkeni sadece bir nevi gizleyerek kazara erişimleri engeller. Bu tür özel değişkenlerin değerlerini okumak veya değiştirmek için doğru yaklaşım, her zaman sınıf içindeki metotları kullanmaktır.

SIRA SİZDE

Öğrencilerin bir sınavdan aldığı puanları girerek sınıf ortalamasının hesaplandığı bir program yazdığınızı düşününüz. Yanlışlıkla puan olarak 100'den büyük bir değer girerseniz ne olur? Programa hatalı veri girişini nasıl engelleyebileceğinizi tartışınız ve bu bilgilere göre diyagramı doldurunuz.



2. UYGULAMA

Verileri Kapsülleme İle Koruyorum

Süre: 30 dakika

Amaç: Belirlenen temel veriyi koruma altına alan program yazmak

Yönerge: Yazılan adımları takip ederek uygulamayı tamamlayınız.

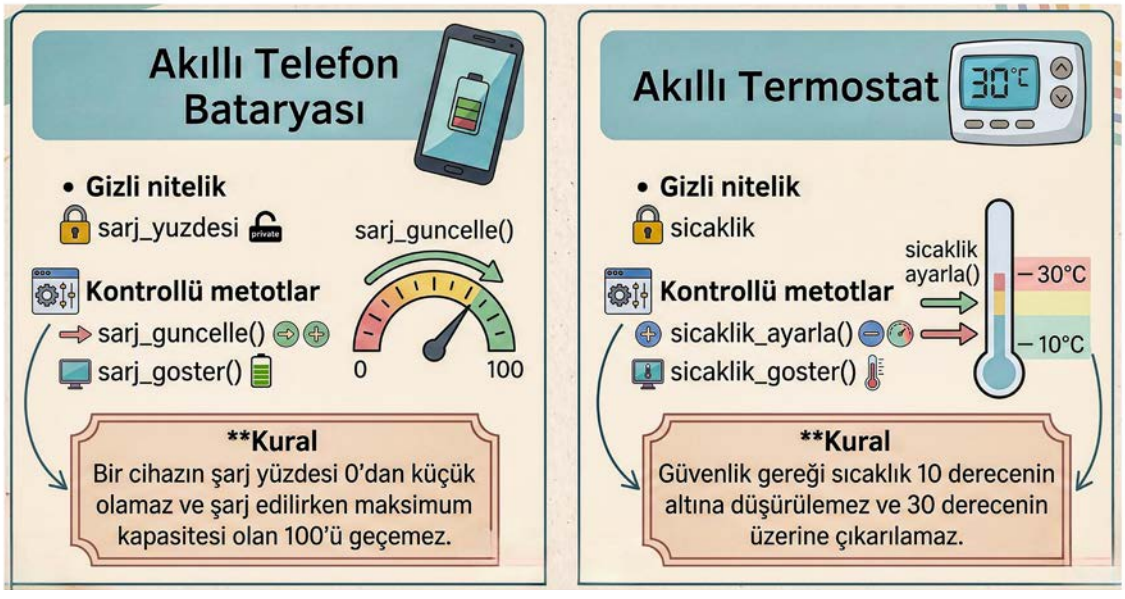
1. Adım: Görsel 7.2'yi inceleyerek hangi örnek sınıf için program yazacağınızı belirleyiniz ve bu sınıfı tanımlayınız.

2. Adım: Tanımladığınız sınıftaki temel niteliği gizli (private) yapınız.

3. Adım: Bu veriyi güvenli bir şekilde okumak veya değiştirmek için kurallar içeren metotları yazınız.

4. Adım: Veriyi değiştiren metotlarınızın içine hatalı veri girişini engelleyecek bir güvenlik şartı (if-else mantığı) ekleyiniz.

5. Adım: Yazdığınız programı çalıştırınız. Doğru ve hatalı örnek veriler girerek programın doğru çalışıp çalışmadığını test ediniz.



Görsel 7.2: Örnek sınıflar

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

7.2. MİRAS ALMA İŞLEMLERİ

Bir sınıfın başka bir sınıfın özellikleri ve metotlarını devralmasına imkân veren nesne yönelimli programlama ilkesine **miras alma** (inheritance) denir. Böylece kod tekrarı azalır, programın bakımı kolaylaşır ve sınıflar arasında hiyerarşik bir yapı kurulur.

Hayvanlar âleminde tüm hayvanlar nefes alır, hareket eder ve uyur. Köpekler de bu özelliklere sahiptir ve havlamak, kuyruk sallamak gibi kendi tabiatına uygun davranışları da sergiler. Kangal cinsi köpek tüm bu özellikleri göstermekle birlikte kendine özgü niteliklere de sahiptir. Bu bilgiden hareketle "Tüm kangallar köpektir ancak tüm köpekler kangal değildir." cümlesi, programlamada bir alt varlığın üst varlığa ait özellikleri miras almasına örnek olarak gösterilebilir. Tablo 7.2'de miras alma ile ilgili kavramlar açıklanmaktadır.

Tablo 7.2: Miras Alma Kavramları

Miras Alma Kavramları	Açıklama	Örnek
Üst Sınıf (Parent Class)	Özelliklerini devreden genel sınıf	Araba
Alt Sınıf (Child Class)	Üst sınıftan miras alan özelleşmiş sınıf	ElektrikliAraba
super()	Alt sınıftan üst sınıfın metodunu çağırır.	super().__init__(...)
Metot Ezme (Override)	Üst sınıf metodunu alt sınıfta yeniden tanımlar.	yakit_doldur()

Alt Sınıf Oluşturma

Alt sınıf tanımlanırken sınıf adının yanına parantez içinde üst sınıfın adı yazılır. Alt sınıfın __init__() metodu içinde super().__init__() çağrılarak üst sınıfın nitelikleri **devralınır.**

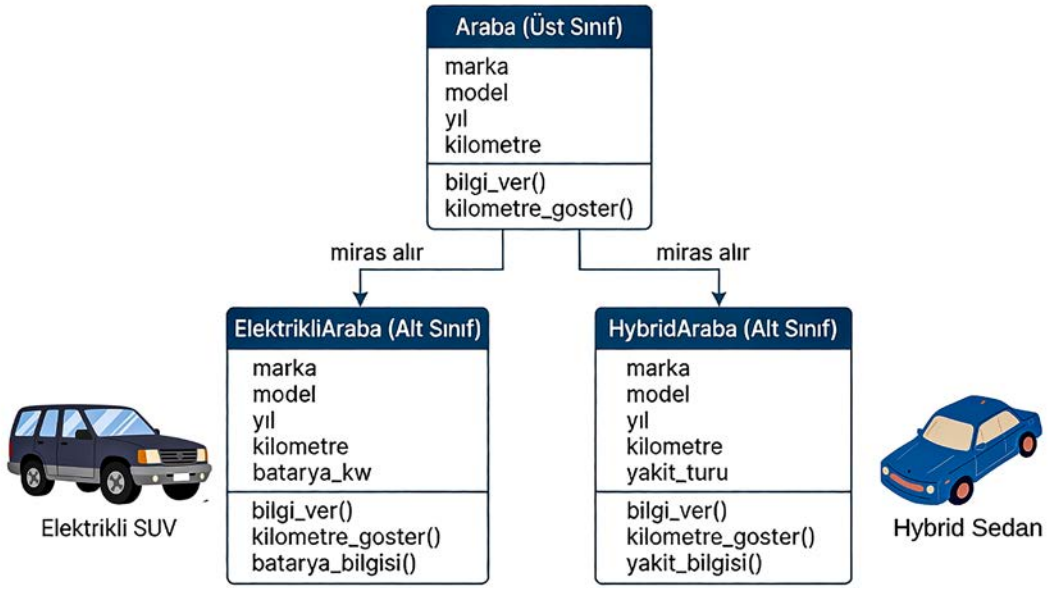
5. ÖRNEK UYGULAMA

Amaç: Miras yoluyla bir **alt sınıf üreten** program yazmak

1. Adım: Görsel 7.3 incelenir. Bu görsele göre bir **Araba** üst sınıfı oluşturulur.

2. Adım: Araba üst sınıfından **ElektrikliAraba** alt sınıfı oluşturulur.

3. Adım: Araba ve ElektrikliAraba sınıflarından birer nesne oluşturularak özellikleri ekrana yazdırılır.



Görsel 7.3: Araba üst sınıfı ve bu sınıftan üretilen alt sınıflar

```python
# Üst sınıf (parent class)
class Araba:
    #Bir arabayı temsil eden üst sınıf.
    def __init__(self, marka, model, yil):
        self.marka  = marka
        self.model  = model
        self.yil    = yil
        self.kilometre = 0
    def bilgi_ver(self):
        print(f"{self.yil} {self.marka} {self.model}")
    def kilometre_goster(self):
        print(f"Kilometre: {self.kilometre} km")

# Alt sınıf (child class)
class ElektrikliAraba(Araba):           # Araba'dan miras alır
    def __init__(self, marka, model, yil, batarya_kw=75):
        super().__init__(marka, model, yil)  # Üst sınıfı başlat
        self.batarya_kw = batarya_kw         # Alt sınıfa özgü nitelik

    def batarya_bilgisi(self):
        print(f"Batarya kapasitesi: {self.batarya_kw} kWh")

araba     = Araba('Marka1', 'Hibrit', 2022)
e_araba   = ElektrikliAraba('Marka2', 'SUV', 2025)

araba.bilgi_ver()
e_araba.bilgi_ver()
e_araba.batarya_bilgisi()
```

EKRAN ÇIKTISI
2022 Marka1 Hibrit
2025 Marka2 SUV
Batarya kapasitesi: 75 kWh

SIRA SİZDE

Görsel 7.3'ü inceleyiniz. **HybridAraba** alt sınıfını tanımlayarak bilgisayarda kodlayınız.

BİLGİ KUTUSU

super() işlevi, alt sınıfın üst sınıfa ait herhangi bir metoda erişmesini sağlar. Bu işlevin en yaygın kullanımı, alt sınıfın __init__() metodunda üst sınıfın niteliklerini başlatmak için super().__init__(...) çağrısı yapmaktır. Bu sayede üst sınıfta tanımlı tüm nitelikler, alt sınıfın nesnelerinde de kullanılabilir.

Alt Sınıfa Yeni Özellik ve Metot Ekleme

Alt sınıf üst sınıftan devraldığı özellik ve metotlara ek olarak kendine özgü nitelikler ve metotlar içerebilir. Bu yöntem, genel yapıyı korurken sınıfı ihtiyaca göre özelleştirir.

6. ÖRNEK UYGULAMA

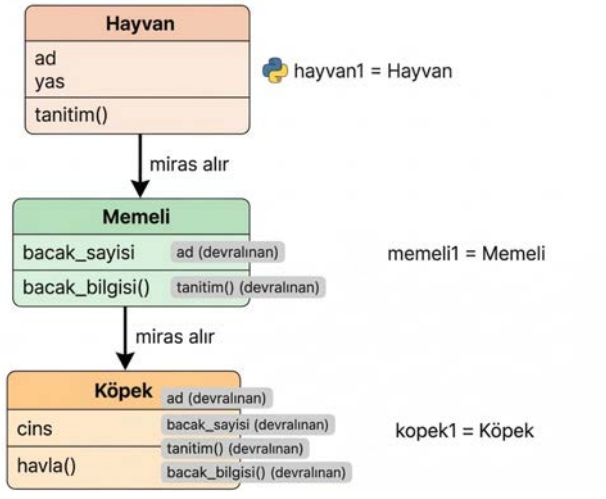
Amaç: Üst sınıf, ara sınıf ve alt sınıf tanımlamak

1. Adım: Görsel 7.4 incelenir. Bu görsele göre bir **Hayvan** üst sınıfı oluşturulur.

2. Adım: Özellik ve metotları belirtilen **Memeli** ara sınıfı oluşturulur.

3. Adım: Özellik ve metotları belirtilen **Kopek** alt sınıfı oluşturulur.

4. Adım: Karabaş adında üç yaşında bir kangal köpek tanımlanarak özellikleri ekrana yazdırılır.



Görsel 7.4: Birden fazla katmanlı miras alma yapısı

```
# 1. katman - Genel üst sınıf
class Hayvan:
    def __init__(self, ad, yas):
        self.ad  = ad
        self.yas = yas
    def tanitim(self):
        print(f"Ad: {self.ad}, Yaş: {self.yas}")
# 2. katman - Ara sınıf
class Memeli(Hayvan):
    def __init__(self, ad, yas, bacak_sayisi):
        super().__init__(ad, yas)
        self.bacak_sayisi = bacak_sayisi
    def bacak_bilgisi(self):
        print(f"Bacak sayısı: {self.bacak_sayisi}")
# 3. katman - En özelleşmiş alt sınıf
class Kopek(Memeli):
    def __init__(self, ad, yas, cins):
        super().__init__(ad, yas, 4)  # Köpeklerde bacak sayısı sabit 4 olarak gönderiliyor
        self.cins = cins
    def havla(self):
        print(f"{self.ad} havlıyor: Hav hav!")

# Test
kopek1 = Kopek('Karabaş', 3, 'Kangal')
kopek1.tanitim()          # Hayvan'dan miras → Ad: Karabaş, Yaş: 3
kopek1.bacak_bilgisi()      # Memeli'den miras → Bacak sayısı: 4
kopek1.havla()            # Kendine özgü  → Karabaş havlıyor: Hav hav!
```

EKRAN ÇIKTISI

```
Ad: Karabaş, Yaş: 3
Bacak sayısı: 4
Karabaş havlıyor: Hav hav!
```

SIRA SİZDE

Miras almanın yazılım geliştirmede sağladığı iki temel avantajı örneklerle açıklayınız.

1. Avantaj:

2. Avantaj:

Metot Ezme

Üst sınıfta tanımlı bir metot, alt sınıf için uygun değilse veya farklı davranması gerekiyorsa alt sınıfta **aynı adla** yeniden tanımlanır. Metodun aynı isimle ancak farklı bir içerikle tekrar tanımlanmasına **metot ezme** (override) denir. Programlama dilinde nesne alt sınıfa ait ise üst sınıfın metodunu yok sayarak alt sınıfın metodunu çalıştırır.

7. ÖRNEK UYGULAMA

Amaç: Üst sınıftan miras yoluyla oluşturulan alt sınıfta aynı metot ismi kullanılarak metodu ezme işlemi yapmak

```
1   # 1. Üst Sınıf (Genel şablon)
2   class Kus:
3       def hareket_et(self):
4           print("Kanat çırparak gökyüzünde uçuyor.")
5
6   # 2. Alt Sınıf (Miras alır ama metodu ezer)
7   class Penguen(Kus):
8       # Üst sınıftaki metot adı (hareket_et) aynen yazılır.
9       # metot ezilerek bu sınıfa ait özel bir davranış tanımlanır.
10      def hareket_et(self):
11          print("Uçamaz! Karnının üstünde buzda kayar ve suda yüzer.")
12  serce=Kus()
13  kral=Penguen()
14
15  print("Sıradan Bir Kuşun Hareketi")
16  serce.hareket_et()
17
18  print("\nPenguenin Hareketi")
19  kral.hareket_et()
```

EKRAN ÇIKTISI

```
Sıradan bir kuşun hareketi
Kanat çırparak gökyüzünde uçuyor.
Penguenin hareketi
Uçamaz! Karnının üstünde buzda kayar ve suda yüzer.
```

DİKKAT

Metot ezme işleminin gerçekleşmesi için alt sınıfta tanımlanan metodun adı, üst sınıftaki metot adıyla tam olarak aynı olmalıdır. Farklı bir ad kullanılırsa ezme değil yeni bir metot oluşturma işlemi gerçekleşir.

3. UYGULAMA

Alt Sınıflara Miras Devreden Program Yazıyorum

Süre: 40 dakika

Amaç: Sınıf hiyerarşisi tasarlayarak miras alma işlemini yapan bir program yazmak

Yönerge: Yazılan adımları takip ederek uygulamayı tamamlayınız.

1. Adım: Verilen gerçek dünya senaryolarını inceleyerek birini seçiniz.

- taşıt → kara taşıtı → otobüs
- çalışan → öğretmen → branş öğretmeni
- elektronik cihaz → akıllı cihaz → akıllı telefon

2. Adım: Her sınıf düzeyi için en az iki nitelik ve bir metot belirleyerek not ediniz.

3. Adım: Üst, alt ve en alt sınıf hiyerarşisini mantıksal olarak doğru şekilde kurgulayıp kodlayınız.

4. Adım: Üst sınıf tasarımında, ortak özellik ve metotları doğru belirlediğinizden emin olunuz.

5. Adım: Alt sınıfların, üst sınıftan kalıtımı doğru olarak aldığından emin olunuz.

6. Adım: Alt sınıflarda en az birer metot tanımladığınızdan ve üst sınıftan gelen metodu özelleştirdiğinden emin olunuz.

7. Adım: Tanımladığınız en alt sınıftan bir nesne oluşturarak örnek özellik değerleri ile metotları çalıştırıp programınızı test ediniz.

8. Adım: Yazdığınız programı dijital portfolyonuza kaydediniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

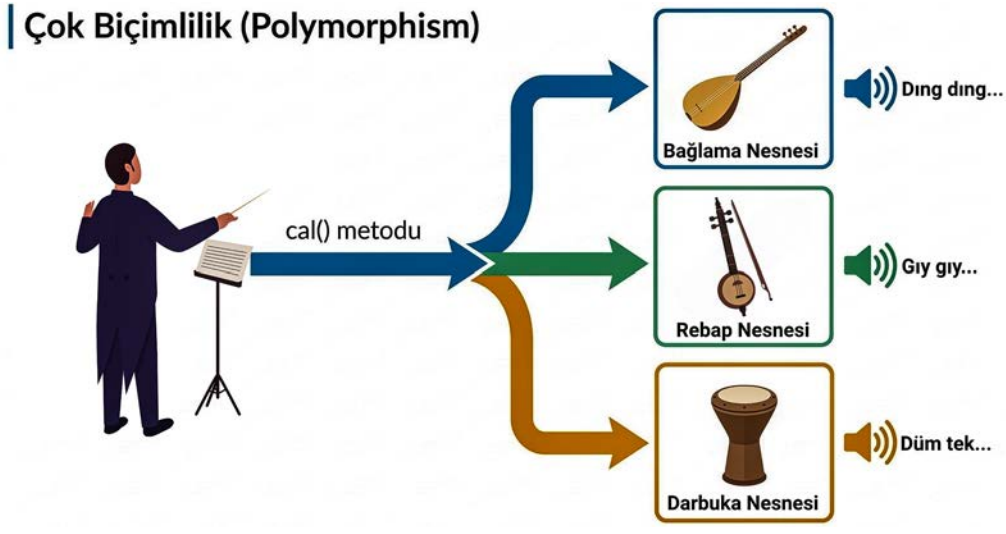
Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

7.3. ÇOK BİÇİMLİ FONKSİYONLAR

Farklı sınıflara ait nesnelerin aynı isimle tanımlanan metodu, kendi yapılarına uygun biçimde gerçekleştirmesine **çok biçimlilik** (polymorphism) denir. Bu sayede farklı türden nesneler aynı arayüz (metot adı) üzerinden yönetilebilir ve hangi nesnenin hangi davranışı sergileyeceğine program çalışma zamanında karar verilir.

Örneğin "çal" komutu verildiğinde bağlama, rebap ve darbuka farklı sesler çıkarır. Üç enstrümanın ortak bir eylemi (çalmak) vardır ancak bu eylemi gerçekleştirme biçimleri birbirinden farklıdır.

Üç enstrümanın "çal" eylemini çok biçimli fonksiyonlar ile aynı anda tetiklemek mümkündür (Görsel 7.5).



Görsel 7.5: Tek fonksiyonun çok biçimliliği

Çok biçimliliğin en yaygın uygulaması, üst sınıfta tanımlanan bir metodun alt sınıflarda ezilerek farklı biçimlerde gerçekleştirilmesidir. Her alt sınıf, ortak metot adını kendi yapısına uygun içerikle doldurur (Tablo 7.3).

Tablo 7.3: Çok Biçimli Fonksiyonların Özellikleri

Özellik	Açıklama	Örnek
Ortak Metot Adı	Farklı sınıflarda aynı isimli metot tanımlanır.	cal()
Farklı Davranış	Her sınıf metodu kendine özgü biçimde uygular.	Bağlama → 'dıng dıng', rebap → 'gıy gıy', darbuka → 'düm tek'
Ortak Arayüz	Farklı nesneler tek bir döngü ile yönetilebilir.	enstruman.ses_cikar()

BİLGİ KUTUSU

Çok biçimlilik kullanılmayan programlarda her yeni nesne için uzun **if-else** veya **switch-case** blokları yazmak gerekir. Çok biçimlilikle bu karmaşa ortadan kalkar. Sisteme yeni bir enstrüman eklendiğinde ana kodda hiçbir değişiklik yapılmaz. Kodu kolayca genişletilebilmek için yeni bir sınıf oluşturmak ve **cal()** metodunu **ezmek** yeterlidir.

8. ÖRNEK UYGULAMA

Amaç: Çok biçimli fonksiyon yazmak

1. Adım: Sekil üst sınıfı oluşturulur.

2. Adım: Daire, Dikdortgen ve Ucgen alt sınıfları tanımlanır.

3. Adım: Her alt sınıfın kendi alan hesaplama metodu tanımlanır.

4. Adım: Üst sınıftaki hesaplama metodunu ezme yöntemi ile çok biçimlilik kullanılır.

```
# Üst sınıf - ortak arayüzü tanımlar
class Sekil:
    #Geometrik şekillerin üst sınıfı.
    def __init__(self, renk):
        self.renk = renk
    def alan_hesapla(self):
        #Alt sınıflar bu metodu ezmek zorundadır.
        return 0
    def bilgi_ver(self):
        print(f"{self.renk} renkli {self.__class__.__name__}: Alan = {self.alan_hesapla():.2f}")
# Alt sınıflar-ortak metodu kendine özgü biçimde gerçekleştirir
import math
class Daire(Sekil):
    def __init__(self, renk, yaricap):
        super().__init__(renk)
        self.yaricap = yaricap
    def alan_hesapla(self):          # Eziyor (override)
        return math.pi * self.yaricap ** 2
class Dikdortgen(Sekil):
    def __init__(self, renk, en, boy):
        super().__init__(renk)
        self.en  = en
        self.boy = boy
    def alan_hesapla(self):          # Eziyor (override)
        return self.en * self.boy
class Ucgen(Sekil):
    def __init__(self, renk, taban, yukseklik):
        super().__init__(renk)
        self.taban      = taban
        self.yukseklik  = yukseklik
    def alan_hesapla(self):          # Eziyor (override)
        return 0.5 * self.taban * self.yukseklik
sekiller = [
    Daire('Kırmızı', 5),
    Dikdortgen('Mavi', 4, 6),
    Ucgen('Yeşil', 3, 8)
]
for s in sekiller:
    s.bilgi_ver()
```

EKRAN ÇIKTISI

Kırmızı Renkli Daire: Alan = 78.54

Mavi Renkli Dikdortgen: Alan = 24.00

Yeşil Renkli Ucgen: Alan = 12.00

DİKKAT

Çok biçimliliğin çalışabilmesi için döngü içinde çağrılan metot adının tüm sınıflarda aynı olması gerekir. Bir sınıfta metot adı farklı yazılırsa programlama dili o nesne için **AttributeError** hatası verir.

4. UYGULAMA

Çok Biçimli Fonksiyon İle Uygulama Geliştiriyorum

Süre: 40 dakika

Amaç: Çok biçimli fonksiyon kullanmak

Yönerge: Yazılan adımları takip ederek uygulamayı tamamlayınız.

1. Adım: Verilen gerçek dünya senaryolarını inceleyerek birini seçiniz.

- Farklı ödeme yöntemleri (nakit, kredi kartı, EFT) → ortak metot: odeme_yap()
- Farklı araç türleri (otomobil, bisiklet, tekne) → ortak metot: hareket_et()
- Farklı bildirim kanalları (e-posta, SMS, uygulama içi) → ortak metot: bildirim_gonder()

2. Adım: Seçtiğiniz senaryoya göre bir üst sınıf oluşturarak ortak metodunuzu tanımlayınız.

3. Adım: Her biri kendi alt metoduna sahip olan en az üç farklı alt sınıf tanımlayınız.

4. Adım: Alt sınıflardan oluşturduğunuz nesneleri bir döngü içerisinde ortak metot üzerinden çağırarak ekrana listeleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

7.4. SOYUT SINIFLAR

Doğrudan nesne üretilemeyen yalnızca alt sınıflara şablon görevi üstlenen sınıf türüne **soyut sınıf** (abstract class) denir. Alt sınıfların metotlarını içeren şablonu tanımlar. Programlama dilinde soyut sınıflar **abc** (abstract base classes) modülü ile tanımlanır.

Karmaşık bir sistemin yalnızca dış davranışını ortaya koymak, iç ayrıntıları gizlemek **soyutlama** (abstraction) olarak adlandırılır. Bir arabayı kullanmak için motorun nasıl çalıştığını bilmek gerekmez. Gaz ve fren pedallarının ne işe yaradığını bilmek yeterlidir. Benzer biçimde soyut sınıf, ne yapılması gerektiğini tanımlar, nasıl yapılacağını alt sınıflara bırakır.

Soyut metotlar **@abstractmethod** dekoratörü ile işaretlenir. Bu metotların gövdesi boş bırakılabilir ya da **pass** komutu ile geçilir. Herhangi bir soyut metodu gerçekleştirmeyen alt sınıftan nesne oluşturulmak istenirse program hata verir. Soyut sınıf kavramları Tablo 7.4'te listelenmiştir.

Tablo 7.4: Soyut Sınıf Kavramları

Kavram	Açıklama	Program Karşılığı
abc Modülü	Soyut sınıf altyapısını sağlayan program modülü	from abc import ABC, abstractmethod
Soyut Sınıf	Nesne üretilemeyen şablon sınıf	ABC'den türeyen sınıf
Soyut Metot	Gövdesi olmayan, alt sınıfta doldurulması zorunlu metot	@abstractmethod
Somut Alt Sınıf	Tüm soyut metotları gerçekleştiren sınıf	class X(SoyutSinif):

9. ÖRNEK UYGULAMA

Amaç: Soyut sınıftan doğrudan bir nesne oluşturulmaya çalışıldığında ne olacağını gösteren bir program yazmak

```
1   # abc modülünden gerekli araçlar içe aktarılıyor
2   from abc import ABC, abstractmethod
3
4   # Soyut sınıf — ABC'den türetiliyor
5   class Sekil(ABC):
6       #Tüm geometrik şekillerin soyut şablonu.
7       def __init__(self, renk):
8           self.renk = renk
9       @abstractmethod
10      def alan_hesapla(self):
11          #Alt sınıflar bu metodu mutlaka gerçekleştirmelidir.
12          pass
13      @abstractmethod
14      def cevre_hesapla(self):
15          #Alt sınıflar bu metodu mutlaka gerçekleştirmelidir.
16          pass
17      def bilgi_ver(self):                # Soyut olmayan (normal) metot
18          print(f"Renk  : {self.renk}")
19          print(f"Alan  : {self.alan_hesapla():.2f} cm²")
20          print(f"Çevre : {self.cevre_hesapla():.2f} cm")
21  # Soyut sınıftan doğrudan nesne oluşturulamaz!
22  s = Sekil('kırmızı')   #→  TypeError: Can't instantiate abstract class
```

EKRAN ÇIKTISI

```
s = Sekil('kırmızı')   #⟶TypeError: Can't instantiate abstract class
 ~~~~~^^^^^^^^^^
TypeError: Can't instantiate abstract class Sekil without an implementation for abstract methods 'alan_hesapla', 'cevre_hesapla'
```

Programın hata vermesinin nedeni, şeklin türünün tanımlanmamış olmasıdır. Detayı belirtilmemiş bir şeklin alanı ve çevresi hesaplanamaz.

BİLGİ KUTUSU

@abstractmethod dekoratörü bir metodun soyut olduğunu programa bildirir. Bu dekoratörle işaretlenen metot, alt sınıfta yeniden tanımlanmak zorundadır. Aksi durumda program nesne oluşturma adımında TypeError hatası verir. Dekoratör, metodun üst satırına @abstractmethod biçiminde yazılır.

Soyut sınıflar büyük yazılım projelerinde ekip içi iş bölümünü standartlaştırmak için kullanılır. Soyut sınıf, geliştiriciler arasında **ortak bir sözleşme** işlevi görür. Her geliştirici kendi alt sınıfını bu sözleşmeye uygun biçimde yazar.

10. ÖRNEK UYGULAMA

Amaç: Soyut sınıf yapısını kullanan bir program yazmak

1. Adım: OdemeSistemi soyut sınıfı oluşturulur.

2. Adım: Kredi kartı ve dijital cüzdan ile ödeme yöntemlerini tanımlayan alt sınıflar, ödeme sistemi soyut sınıfından oluşturulur.

3. Adım: Ödemeler bir **liste** içinde işlenerek çalıştırılır.

Etik ve Gizlilik Notu: Finansal uygulamalarda gerçek kredi kartı verileri kullanılmamalıdır. Kredi kartı numarası **********3456 şeklinde maskelenebilir.

```
from abc import ABC, abstractmethod
class OdemeSistemi(ABC): # Soyut sınıf
    #Tüm ödeme yöntemlerinin uyması gereken şablon.
    def __init__(self, kullanici_adi):
        self.kullanici_adi = kullanici_adi
    @abstractmethod
    def odeme_yap(self, tutar):
        #Ödemeyi gerçekleştirir.
        pass
    @abstractmethod
    def islem_dogrula(self):
        #İşlemi doğrular.
        pass
    def makbuz_ver(self, tutar):     # Soyut olmayan metot
        print(f"--- MAKBUZ ---")
        print(f"Kullanıcı : {self.kullanici_adi}")
        print(f"Tutar     : {tutar:,.2f} TL")
        print(f"Yöntem    : {self.__class__.__name__}\n")
class KrediKarti(OdemeSistemi):
    def __init__(self, kullanici_adi, kart_no):
        super().__init__(kullanici_adi)
        self.kart_no = kart_no
    def odeme_yap(self, tutar):
        print(f"Kart ({self.kart_no[-4:]}) ile {tutar:,.2f} TL ödendi.")
        self.makbuz_ver(tutar)
    def islem_dogrula(self):
        return len(self.kart_no) == 16
class DijitalCuzdan(OdemeSistemi):
    def __init__(self, kullanici_adi, bakiye):
        super().__init__(kullanici_adi)
```

```
31          self.bakiye = bakiye
32      def odeme_yap(self, tutar):
33          if self.bakiye >= tutar:
34              self.bakiye -= tutar
35              print(f"Dijital cüzdan ile {tutar:,.2f} TL ödendi.")
36              self.makbuz_ver(tutar)
37          else:
38              print("Yetersiz bakiye!")
39      def islem_dogrula(self):
40          return self.bakiye > 0
41  odemeler = [
42      KrediKarti('A*** K***', '************3456'),
43      DijitalCuzdan('M**** D****', 5000),
44  ]
45  for odeme in odemeler:
46      if odeme.islem_dogrula():
47          odeme.odeme_yap(1250)
```

EKRAN ÇIKTISI

```
Kart (3456) ile 1,250.00 TL ödendi.
--- MAKBUZ ---
Kullanıcı : A*** K***
Tutar     : 1,250.00 TL
Yöntem    : KrediKarti

Dijital cüzdan ile 1,250.00 TL ödendi.
--- MAKBUZ ---
Kullanıcı : M**** D****
Tutar     : 1,250.00 TL
Yöntem    : DijitalCuzdan
```

BİLGİ KUTUSU

Çok biçimlilik işletim sistemindeki **yazdır** komutu gibi düşünülebilir. Paket programlarda ortak bir yazdır komutu bulunur. Her program yazdırma işlemini kendi veri yapısına göre farklı şekillerde gerçekleştirir. İşletim sistemi belgenin türüyle ilgilenmez; sadece o nesnenin yazdir() metodunu çağırır ve diğer işlemleri nesnenin kendisine bırakır.

5. UYGULAMA

Soyut Sınıflar İle Günlük Hayat Problemini Çözüyorum

Süre: 40 dakika

Amaç: Soyut sınıf tasarlamak ve somut sınıf oluşturan program yazmak

Yönerge: Yazılan adımları takip ederek uygulamayı tamamlayınız.

1. Adım: Verilen gerçek dünya senaryolarından birini seçiniz.

- Müzik yayın platformu- ücretsiz, premium ve aile paketleri
- Kargo taşıma sistemi- kara yolu, deniz yolu, hava yolu taşımacılığı
- Oyun karakterleri üretimi- işçi, savaşçı, komutan karakterleri

2. Adım: Kendinize "Tüm türler için ortak olan zorunlu davranışlar nelerdir?" sorusunu sorunuz. Tüm türler için ortak olan zorunlu davranışları belirleyerek soyut sınıf yapısını kurgulayınız.

3. Adım: Tanımladığınız soyut sınıf için ortak ve zorunlu bir metot tanımlayınız.

4. Adım: En az iki adet somut alt sınıf tanımlayınız. Bu sınıflar soyut sınıftan miras almalı ve kendilerine özgü metotlar içermelidir.

5. Adım: Soyut sınıftan doğrudan nesne üretilemeyeceği kuralına uyup uymadığınızı kontrol ediniz.

6. Adım: Programınızı çalıştırarak test ediniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

SIRA SİZDE

Aşağıda nesne yönelimli programlama ilkelerinin günlük hayattaki kullanımı ile ilgili bir metin verilmiştir.

Ünlü bir simit zinciri, Türkiye'de bayilik (franchise) sistemiyle büyümektedir. Markanın merkez ofisi, tüm şubeler için zorunlu bir **Şube Standartları Rehberi** yayımlar. Bu rehberde şubelerin yapması gereken temel işlemler açıkça belirtilmiştir: siparis_al(), standart_simit_servis_et() ve gun_sonu_raporu_ilet(). Ayrıca her şubenin bir **sube_kodu** ve gizli tutulması gereken bir **kasa_sifresi** vardır.

Merkez ofis fiziksel olarak simit satmaz veya müşteri ağırlamaz. Sadece standartları belirleyen bir yönetim birimidir. Gerçek simit satışı, bu rehberin içini kendi fiziksel imkânlarıyla dolduran, (örneğin Kadıköy şubesi veya Kapadokya şubesi) tarafından yapılır. Bir girişimci, rehberdeki kurallardan herhangi birini kendi sistemine entegre etmezse merkez ofis o şubenin faaliyete geçmesine izin vermez. Gurme şubeler gibi bazı özel şubeler, standart servis metodunu değiştirerek simidin yanında özel peynirler ikram edebilir.

Verilen metne göre aşağıdaki soruları cevaplayınız.

1. Merkez ofisin fiziksel bir şube olmaması sadece kuralları belirleyen bir rehber olması programlamadaki hangi yapıya karşılık gelir?
2. Rehberdeki kurallardan birini eksik uygulayan şubenin faaliyetine (programın çalışmasına) izin verilmemesi programda hangi teknik zorunluluğu temsil eder?
3. Her şubenin kasa_şifresi gibi kritik verilerinin dışarıdan doğrudan değiştirilememesi ve sadece merkez ofisin belirlediği güvenli yöntemlerle erişilmesi hangi nesne yönelimli programlama kavramıdır?
4. Gurme şubelerin, standart rehberdeki standart_simit_servis_et() metodunu alıp kendi özel sunumlarıyla değiştirmesi (modifiye etmesi) hangi mekanizma ile açıklanır?
5. Sisteme ileride **Mobil Karavan** veya **Havalimanı Şubesi** gibi yeni konseptler eklendiğinde en baştan bir **Soyut Sınıf** (rehber) kullanılmış olması, yazılımın sürdürülebilirliğine ve hata payının azalmasına nasıl katkı sağlar?

7.1. PROJE	
Konu	Akıllı Kütüphane Sistemi Yazıyorum
Kapsanan Konular	Sınıf ve nesne, kapsülleme, miras alma, çok biçimlilik, soyut sınıflar
Süre	40+40+40 dakika
Amaç	Nesne yönelimli programlama temel kavramlarını kapsayıcı bir projede kullanmak
Çalışma Biçimi	Bireysel

Proje Senaryosu

Bir okul kütüphanesinde ortak ve farklı özelliklere sahip materyallerin (kitap, dergi, sesli kitap) dijital ortamda yönetilmesini sağlamaktır. Kullanıcıların materyalleri kolayca ödünç alıp iade edebileceği, her materyal türünün ödünç bilgisini kendine özgü biçimde oluşturabildiği esnek ve genişletilebilir bir sistem tasarlamaktır.

1. Aşama: Materyal soyut temel sınıfını kodlayınız.

Kapsanan İlke: Soyut Sınıf ve Sınıf-Nesne

Sistemin temelini oluşturan **Materyal** soyut sınıfı verilen gereksinimleri karşılamalıdır.

- isbn (veya kod)
- odunc_bilgisi() -> soyut metot (Her materyal türü kendi metnini üretir.)
- durum_goster() -> soyut metot (Her materyal türü kendi durum bilgisini gösterir.)
- tanitim() -> soyut olmayan metot (Tüm alt sınıflar miras alır; temel bilgileri listeler.)

Bir kitabı tanımlamak için başka hangi özellikler eklenmelidir? En az **üç özelliği** de kendiniz ekleyiniz.

1. Görev

a) **Materyal** sınıfını eksiksiz biçimde kodlayınız.
b) Sınıftan doğrudan nesne oluşturmayı deneyiniz ve aldığınız hata mesajını not ediniz.
c) Hangi metotların soyut, hangilerinin somut (normal) olduğunu ve nedenini açıklayınız.

2. Aşama: Alt Sınıflar: Miras Alma

Kapsanan İlke: Miras Alma · Metot Ezme - super()

Aşağıdaki tabloya göre **materyal** sınıfından türeyen üç somut alt sınıf oluşturunuz.

Alt Sınıf	Ek Nitelikler	Soyut Metot Gerçekleştirmeleri
Kitap	tur (roman, bilim vb.), sayfa_sayisi	odunc_bilgisi(), durum_goster()
Dergi	sayi_no, konu	odunc_bilgisi(), durum_goster()
SesliKitap	sure_dk, anlatici	odunc_bilgisi(), durum_goster()

2. Görev

a) Dergi() ve SesliKitap() sınıflarını yukarıdaki yapıyı örnek alarak kodlayınız.
b) Her alt sınıfın odunc_bilgisi() çıktısını birbirinden farklılaştıran niteliği belirtiniz.
c) super() çağrısını kaldırarak ne olduğunu gözlemleyiniz ve sebebini açıklayınız.

3. Aşama: Kullanıcı Sınıfı ve Ödünç İşlemleri

Kapsanan İlke: Sınıf & Nesne · Metot Tasarımı

Sisteme kayıtlı kütüphane kullanıcılarını temsil eden **Kullanici** sınıfı oluşturulacaktır.

Kullanıcı sınıfının nitelikleri: uye_no, ad_soyad, odunc_listesi (başlangıçta boş liste)

Metotlar

- odunc_al(materyal) → Materyali ödünç listesine ekler, odunc_durumu'nu **True** yapar.
- iade_et(materyal) → Materyali listeden çıkarır, odunc_durumu'nu **False** yapar.
- odunc_listesi_goster() → Kullanıcının ödünçteki tüm materyallerini listeler.

3. Görev

a) Kullanici sınıfını kodlayınız.
b) odunc_al() metodunda neden if not materyal.odunc_durumu kontrolü yapıldığını açıklayınız.
c) iade_et() metoduna materyalin bu kullanıcıda olmadığı durumu için uygun mesaj ekleyiniz.

4. Aşama: Kütüphane Kataloğu: Çok Biçimlilik

Kapsanan İlke: Kapsülleme, Çok Biçimlilik, Ortak Arayüz

Sistemin tüm materyallerini tutan **Katalog** sınıfı, farklı türdeki nesneleri tek bir liste üzerinden yönetir. Bu yapı çok biçimliliğin en somut uygulamasıdır.

4. Görev

a) Katalog sınıfını kodlayınız.
b) tum_materyalleri_listele() metodundaki m.tanitim() çağrısının polimorfik olduğunu açıklayınız.
c) odunc_islemi() adında merkezi bir metot ekleyerek ödünç işlemlerini Katalog sınıfına taşıyınız. 3. aşamada yazılan Kullanici.odunc_al() ve Kullanici.iade_et() metodu Katalog onayıyla çalışacaktır.
ç) Kataloğa en az 2 kitap, 1 dergi, 1 sesli kitap ekleyiniz.
d) Sadece kitapları listeleyen bir kod satırı yazınız (tur_ile_ara() ile).

5. Aşama: Sistemin Bütününü Test Etme

Kapsanan İlke: Tüm OOP ilkeleri entegre edilmiş biçimde

Verilen örnek test senaryosunu **kendi programınıza** göre adım adım kodlayınız. Sisteme en az iki kullanıcı ve farklı türlerde dört materyal eklenmelidir.

Test Senaryosu

```
# 1. Katalog ve kullanıcılar oluşturuluyor
katalog = Katalog()
kullanici1 = Kullanici('U001', 'Ayşe Kaya')
kullanici2 = Kullanici('U002', 'Mehmet Demir')

# 2. Materyaller oluşturuluyor
k1 = Kitap('6254080826', 'Safahat', 'M.Akif Ersoy', 1933, 'Şiir', 784)
k2 = Kitap('978-975-002', 'Tutunamayanlar', 'Oğuz Atay', 1971, 'Roman', 724)
d1 = Dergi('DRG-001', 'Bilim ve Teknik', 'TÜBİTAK', 2024, 692, 'Bilim')
s1 = SesliKitap('SES-001', 'Kürk Mantolu Madonna', 'Sabahattin Ali', 1943, 180, 'Ahmet Yılmaz')

# 3. Materyaller kataloğa ekleniyor
for m in [k1, k2, d1, s1]:
    katalog.ekle(m)

# 4. Katalog görüntüleniyor
katalog.tum_materyalleri_listele()

# 5. Ödünç işlemleri
kullanici1.odunc_al(k1)
kullanici1.odunc_al(s1)
kullanici2.odunc_al(k1)    # Zaten ödünçte - uyarı vermeli

# 6. Ödünç listeleri
kullanici1.odunc_listesi_goster()

# 7. İade ve tekrar kontrol
kullanici1.iade_et(k1)
kullanici2.odunc_al(k1)    # Artık alınabilmeli

# 8. Rafta olanları listele
katalog.rafta_olanlar()
```

5. Görev

a) Test senaryosunu çalıştırınız ve tüm çıktıları kaydediniz.

b) Herhangi bir adımda hata alındıysa hatanın nedenini açıklayarak düzeltiniz.

c) İyi tasarlanmış bir nesneye yönelik sistem, yeni özellikler eklenmesine (gelişime) açık ancak mevcut kodların değiştirilmesine kapalı olmalıdır. Sistemin bu esnekliğini test etmek için eKitap adında yeni bir materyal türü (alt sınıf) oluşturunuz. Bu sınıfa ek nitelik olarak format (PDF/EPUB) ve dosya_boyutu (MB) ekleyiniz. Bu yeni sınıfı sisteme dahil edip kataloğa eklerken daha önce yazdığınız **Katalog** veya temel **Materyal** sınıflarının kodlarında **hiçbir değişiklik yapmamalısınız.**

Değerlendirme

a) Proje çalışmanız öğretmeniniz tarafından **Proje Derecelendirme Ölçeği**'nde yer alan ölçütlere göre değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

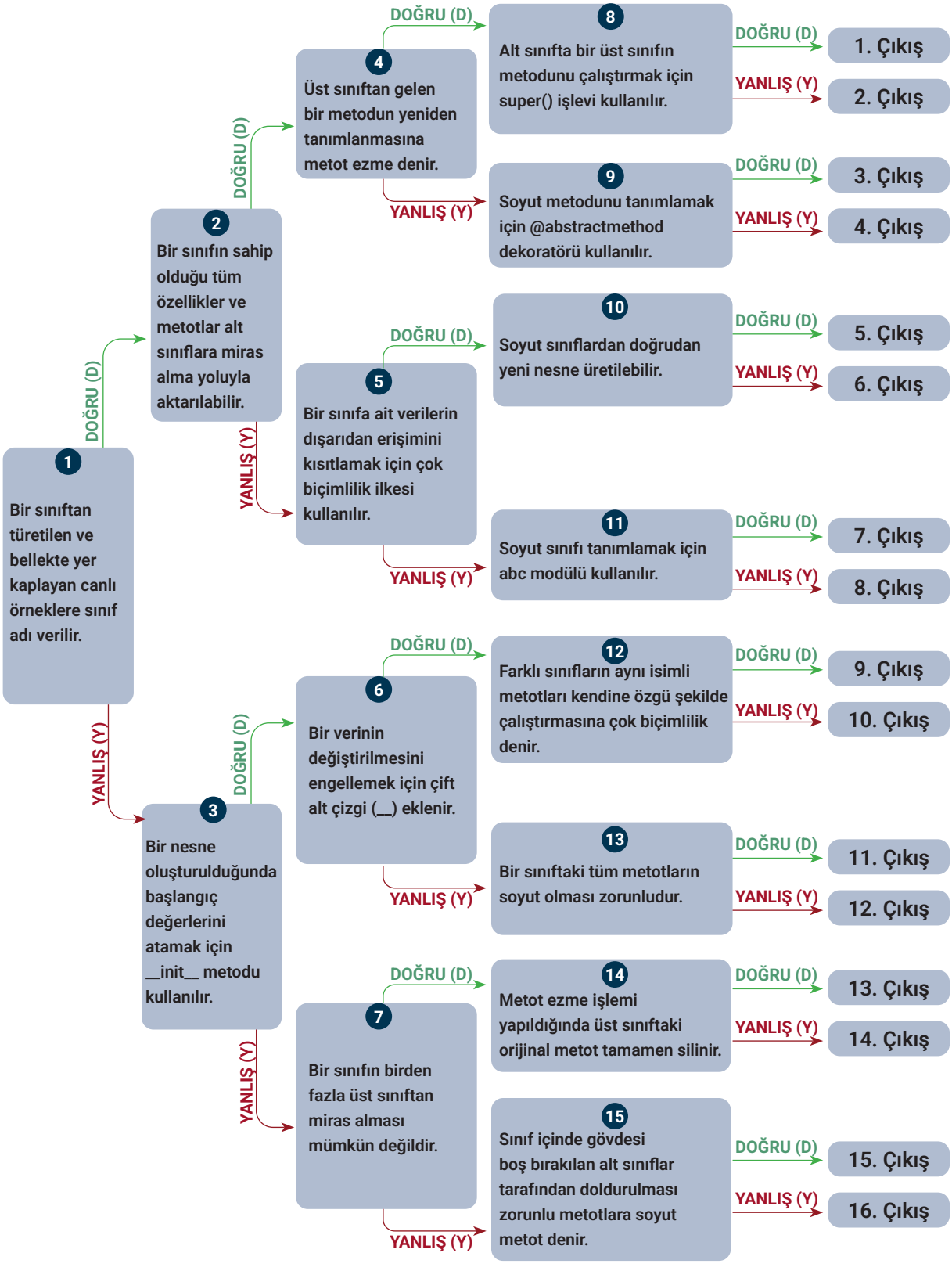
Proje derecelendirme ölçeğine ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

b) **Öz Değerlendirme Formu** proje geliştirme sürecinin sonunda kendinizi değerlendirmeniz amacıyla hazırlanmıştır. Cevaplarınızı tabloda ilgili kutucuğa yazınız. Verilen karekodu kullanarak öz değerlendirme formuna ulaşabilirsiniz.

Öz değerlendirme formuna ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

ÖLÇME VE DEĞERLENDİRME

A) Aşağıda Doğru / Yanlış önermeleri içeren tanılayıcı dallanmış ağaç verilmiştir. Verilen önermenin doğru olduğunu düşünüyorsanız (D), yanlış olduğunu düşünüyorsanız (Y) seçeneğini işaretleyiniz. İlk önermeden başlayarak yapacağınız seçimlerle ve oklar yönünde ilerleyerek bir çıkış noktası belirleyiniz.



B) Aşağıdaki yapılandırılmış gridde numaralandırılmış kutucuklarda nesne yönelimli programlama ile ilgili bazı ifadeler verilmiştir. Kutucuk numaralarını kullanarak soruları cevaplayınız. Aynı kutucuk numarasını birden fazla cevap için kullanabilirsiniz.

(1) abc	(2) sınıf	(3) nesne
(4) super()	(5) çok biçimlilik	(6) miras alma
(7) kapsülleme	(8) @abstractmethod	(9) metot ezme

1. Nesne yönelimli programlama yaklaşımının temelini oluşturan, yazılım mimarisinin güvenliğini, esnekliğini ve yeniden kullanılabilirliğini inşa eden **temel programlama ilkeleri** hangileridir? (.....)

2. Tasarlanan genel bir şablon mimarisinde bellek üzerinde somut ve çalışan dinamik yapılar türetilirken kullanılan, aralarında **şablon örnek ilişkisi** bulunan temel kavramlar hangileridir? (.....)

3. Bir alt sınıfın üst sınıftan tüm nitelik ve işlevleri devralmasını sağlayan mekanizma ile **(__init__)** metoduna doğrudan erişip tetiklemek için kullanılan işlevler hangileridir? (.....)

4. Nesnelerin iç detaylarını, değişkenlerini ve dış dünyadan gizleyerek verilerin yetkisiz değiştirilmesini engelleyen koruma mekanizması hangisidir? (.....)

5. Taslak mimariler (soyut sınıflar) oluşturmak ve alt sınıfların zorunlu olarak doldurması gereken gövdesiz metotları işaretlemek için **abc modülünden** birlikte içe aktarılması gereken bileşenler hangileridir? (.....)

6. Aynı isme sahip bir metodun, nesne türüne bağlı olarak tamamen farklı çıktılar üretebilmesi yeteneğine ne ad verilir? (.....)

7. Alt sınıfların kendilerine has özelliklerini çalıştırırken, miras aldıkları üst sınıftaki aynı isimli metotları geçersiz kılması **kavram** ve **yöntemleri** hangileridir? (.....)

8. Bir yazılım projesinde kod tekrarını sıfıra indirerek **farklı nesnelerin aynı davranışı sergilemesini** sağlamak için birlikte çalışması gereken ana ilkeler hangileridir? (.....)

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Bir alt sınıfta super().__init__() çağrısı yapılmazsa ne olur?

A) Program hata vermeden çalışır, üst sınıf nitelikleri otomatik devralınır.
B) Üst sınıfın nitelikleri alt sınıf nesnesinde tanımsız kalır.
C) Alt sınıf niteliklerine üst sınıftan erişilemez.
D) Alt sınıf otomatik olarak soyut sınıfa dönüşür.
E) Program, super() çağrısını arka planda kendiliğinden tamamlar.

2.

```
class Araba:
    def __init__(self, marka, yil):
        self.marka = marka
        self.yil   = yil
    def bilgi(self):
        print(f"{self.yil} {self.marka}")
a = Araba('Marka1', 2023)
a.bilgi()
```

Verilen komut dizisi çalıştırıldığında aşağıdaki hangi işlem uygulanır?

A) Marka1 2023
B) 2023 Marka1
C) Araba Marka1
D) Araba Marka1 2023
E) Hata mesajı üretilir

3.

I. Miras Alma (Inheritance)

II. Döngüsel Bağımlılık

III. Çok Biçimlilik (Polymorphism)

IV. Soyutlama (Abstraction)

Yukarıdaki kavramlardan hangileri nesne yönelimli programlamanın temel ilkeleri arasında yer alır?

A) Yalnız I
B) I ve II
C) I ve III
D) I, III ve IV
E) I, II, III ve IV

4. Soyut sınıflarla ilgili verilen ifadelerden hangisi yanlıştır?

A) Soyut sınıf yalnızca soyut metotlardan oluşmak zorundadır.
B) Soyut sınıfların içindeki soyut metotları belirtmek için @abstractmethod dekoratörü kullanılır.
C) Soyut sınıftan doğrudan nesne oluşturulamaz.
D) Alt sınıf tüm soyut metotları gerçekleştirmek zorundadır.
E) Soyut sınıf tanımlamak için abc modülü kullanılır.

5.

```
class Hayvan:
    def ses(self): return '...'
class Kedi(Hayvan):
    def ses(self): return 'Miyav'
class Kopek(Hayvan):
    def ses(self): return 'Hav'
for h in [Kedi(), Kopek(), Hayvan()]:
    print(h.ses())
```

Hayvan sınıfı standart bir sınıftır. Verilen komut dizisi çalıştırıldığında aşağıdaki hangi işlem uygulanır?

A) ...
Miyav
Hav

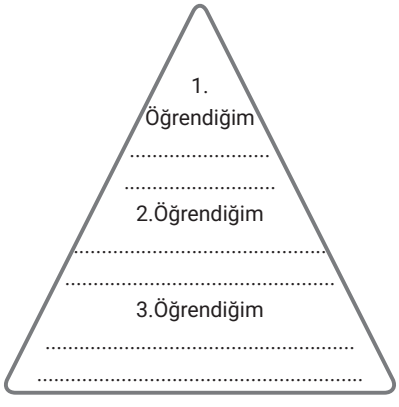
B) Miyav
Hav
...

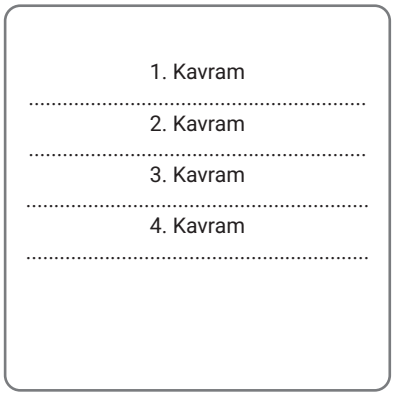
C) Miyav
Hav
Hata

D) ...
...
...

E) Program hatalıdır.

Ç) Nesne yönelimli programlama hakkında öğrendiğiniz üç bilgiyi üçgenin, bu öğrenme biriminde ilk defa karşılaştığınız dört temel kavramı karenin, aklınıza takılan bir soruyu çemberin içine yazınız.









KONULAR

8.1. ARAMA ALGORİTMALARI

8.2. SIRALAMA ALGORİTMALARI

ANAHTAR KAVRAMLAR

- arama algoritması
- doğrusal arama
- ikili arama
- kabarcık sıralama
- seçmeli sıralama
- sıralama algoritması
- sıralı liste



8. ÖĞRENME BİRİMİ

ARAMA VE SIRALAMA ALGORİTMALARI

BEKLENTİLER

Bu öğrenme biriminde sizden

- arama algoritmalarını uygulamanız
- sıralama algoritmalarını uygulamanız

beklenmektedir.

HAZIRLIK ÇALIŞMALARI

1. Aşağıda bir okul kütüphanesinde yaşananlarla ilgili bir metin verilmiştir.

Mert okul kütüphanesindeki kitapların düzenlenmesine yardım eder ancak kütüphanenin iade rafındaki yüzlerce kitap herhangi bir kural gözetilmeden üst üste dizilmiştir. Bir arkadaşı Mert'ten **Yapay Zekâya Giriş** kitabını bulmasını istediğinde, Mert rafın en başından başlayarak kitapları tek tek kontrol etmek zorunda kalır. Bu durumdan dolayı Mert, aradığı kitabı bulmakta zorlanır hatta bazen kitabın rafta olup olmadığından emin olamaz. Kütüphane sorumlusu bu sorunu çözmek için tüm kitapları adlarına göre alfabetik sırayla dizebileceklerini söyler.

Verilen metne göre aşağıdaki soruları cevaplayınız.

a) Mert, arkadaşının istediği kitabı bulmak için nasıl bir yöntem izlemiştir?

b) Kütüphane sorumlusunun önerisi, aranan kitapları bulma sorununa nasıl çözüm getirebilir?

2. Bir önceki öğrenme biriminde nesneleri (kitapları) özellikleriyle (yazar, ad, tür) tanımlamayı öğrendiniz. Sizce binlerce kitap olduğunda aranan kitaba hızlıca ulaşmak için sadece nesne yapısını kullanmak yeterli midir? Neden?

BAŞLARKEN

Bilgisayar parçaları satan bir donanım mağazasının raflarında yüzlerce çeşit ürün bulunur. Her ürün grubunun kendi alt grupları vardır. Bir müşteri 32 GB 3200 MHz DDR4 SODIMM RAM istediğinde bellek modülünün hangi rafta bulunduğu, DDR4 SODIMM belleklerin hangi grupta yer aldığı ve 32 GB kapasitesinin hangi bölümde bulunduğu düzenli bir şekilde kaydedilmelidir. Raflar dağınık durumdaysa bir bellek modülünü bulmak için her rafa tek tek bakılmalıdır. Bu işlem hem uzun zaman alır hem de yorucudur (Görsel 8.1).



Görsel 8.1: Dağınık ve düzenli yerleştirilmiş ürün rafları

Veriler üzerinde arama yapmak, bilgisayar programlarının en temel ihtiyaçlarından biridir. Bir müzik uygulamasında şarkı arama, bir alışveriş sitesinde ürün filtreleme özelliği veya bir rehberde isim arama gibi işlevlerin arka planında arama algoritması çalışır. Bu algoritmalar ne kadar verimli tasarlanırsa program o kadar hızlı ve kullanışlı olur.

8.1. ARAMA ALGORİTMALARI

Bir veri kümesi içinde belirli bir değeri (hedef elemanı) bulmak için izlenen adım adım talimatlara **arama algoritması** (search algorithm) denir. Kullanılacak arama yöntemi verilerin düzenlenme biçimine ve büyüklüğüne göre farklılık gösterir. Arama algoritmalarında kullanılan kavramlar Tablo 8.1'de listelenmiştir.

Tablo 8.1: Arama Algoritması Kavram ve Açıklamaları

Kavram	Açıklama
Veri Kümesi (Liste)	Arama yapılacak eleman dizisi
Hedef Eleman	Listede aranacak değer
İndeks	Listenin konumunu gösterir (0'dan başlar)
Adım Sayısı	Sonuca ulaşmak için gerçekleştirilen karşılaştırma sayısı

Doğrusal Arama

Listenin en başından başlayarak her elemanı sırayla ve hedef değerle karşılaştıran arama yöntemine **doğrusal arama** (lineer search) denir. Hedef eleman bulunduğunda arama sona erer. Listenin sonuna ulaşılmasına rağmen hedef bulunamazsa **bulunamadı** bilgisi döndürülür.

Masanın üzerindeki dağınık kâğıtlar arasında bir belgeyi ararken en üstteki kâğıttan başlayıp tek tek kontrol ederek belgeyi bulmak doğrusal aramaya benzer. Tablo 8.2'de doğrusal arama algoritmasının çalışma mantığı, Tablo 8.3'te **adım adım izleme** (trace) aşamaları verilmiştir.

Tablo 8.2: Doğrusal Arama Algoritması Çalışma Mantığı

Adım	İşlem	Açıklama
1.	İndeks = 0'dan başla	Listenin ilk elemanına git.
2.	liste[indeks] == hedef?	Mevcut elemanı hedefle karşılaştır.
3.a)	Evet ⟶ indeksi döndür	Hedef bulundu, indeksi döndür ve aramayı bitir.
3.b)	Hayır ⟶ indeks += 1	Sonraki elemana geç.
4.	Liste bitti mi?	Evet ⟶ -1 döndür (bulunamadı)

Tablo 8.3: Adım Adım İzleme (liste = [14, 7, 23, 5, 11] aranan = 5)

Adım	İndeks	Karşılaştırılan Eleman	Hedef (5) ile Eşit mi?	Sonuç
1.	0	14	Hayır	Devam et
2.	1	7	Hayır	Devam et
3.	2	23	Hayır	Devam et
4.	3	5	✓ Evet	**İndeks: 3**

BİLGİ KUTUSU

Python, C# ve Java gibi programlama dillerinde listenin ilk elemanı 0. indeksten başlar. 0 ve üzerindeki tüm pozitif sayılar geçerli bir veri konumunu temsil eder. Aranan değerin listede bulunmadığını (hata/yokluk durumunu) sisteme bildirmek için geçerli bir indeks olmayan -1 değeri evrensel bir standart olarak kullanılır.

1. ÖRNEK UYGULAMA

Amaç: Bir sayı listesinde doğrusal arama gerçekleştiren program yazmak

```
1   def dogrusal_arama(liste, hedef):
2       for i in range(len(liste)):
3           if liste[i] == hedef:
4               return i        # hedef bulundu, indeksi döndür
5       return -1               # hedef bulunamadı
6
7   sayilar = [14, 7, 23, 5, 11, 38, 2]
8   aranan  = 11
9
10  sonuc = dogrusal_arama(sayilar, aranan)
11
12  if sonuc != -1:
13      print(f"{aranan} sayısı listede bulundu. İndeks: {sonuc}")
14  else:
15      print(f"{aranan} sayısı listede bulunamadı.")
```

EKRAN ÇIKTISI
11 sayısı listede bulundu. İndeks: 4

SIRA SİZDE

Doğrusal aramaya göre aşağıdaki soruları cevaplandırınız.

1. liste = [9, 3, 17, 42, 6, 28] listesinde doğrusal arama yöntemiyle 42 sayısı aranıyor. Kaç adımda bulunduğunu belirleyiniz. Adım adım izleme tablosunu doldurunuz.

Adım	İndeks	Karşılaştırılan Eleman	Hedef (42) ile Eşit mi?	Sonuç

2. Doğrusal arama yönteminde en fazla kaç karşılaştırma yapılabilir? Bu durumun hangi koşulda gerçekleştiğini açıklayınız.

.....

a) Aranan değer listenin en sonundaysa ne olur?

.....

b) Aranan değer listede hiç yoksa ne olur?

.....

İkili Arama

Yalnızca **sıralı listeler** üzerinde çalışan, her adımda arama aralığını yarıya indirerek hedefe ulaşan algoritmaya **ikili arama** (binary search) denir. Liste ikiye bölünerek hedef değerin, sol yarıda mı sağ yarıda mı olduğuna karar verilir. Böylece sıralı listede çok daha az adımda sonuca ulaşılır.

Alfabetik sıralı bir sözlükte kelime aranırken sözlük ortasından açılır ve kelimenin önce mi sonra mı geldiğine bakılarak diğer yarısı elenir. Tablo 8.4'te ikili arama algoritmasının çalışma mantığı, Tablo 8.5'te izleme aşamaları verilmiştir.

Tablo 8.4: İkili Arama Algoritmasının Çalışma Mantığı

Adım	İşlem	Açıklama
1.	sol=0, sağ=len-1	Arama aralığını tüm liste olarak başlat.
2.	orta = (sol + sağ) // 2	Ortadaki elemanın indeksini hesapla.
3.a)	liste[orta] == hedef	Bulundu ⟶ ortayı döndür.
3.b)	liste[orta] < hedef	Hedef sağ yarıda ⟶ sol = orta + 1
3.c)	liste[orta] > hedef	Hedef sol yarıda ⟶ sağ = orta - 1
4.	sol > sağ mı?	Evet ⟶ -1 döndür (bulunamadı)

Tablo 8.5: Adım Adım İzleme (liste = [2, 5, 9, 14, 20, 31, 45] aranan = 20)

Adım	Sol	Sağ	Orta	liste[orta]	Karar	Sonuç
1.	0	6	3	14	14 < 20 sağa git	sol=4
2.	4	6	5	31	31 > 20 sola git	sağ=4
3.	4	4	4	20	✓20 == 20	İndeks: 4

2. ÖRNEK UYGULAMA

Amaç: Sıralı bir sayı listesinde ikili arama gerçekleştiren programı yazmak

```
1   def ikili_arama(liste, hedef):
2       sol = 0
3       sag = len(liste) - 1
4
5       while sol <= sag:
6           orta = (sol + sag) // 2
7
8           if liste[orta] == hedef:
9               return orta             # hedef bulundu
10          elif liste[orta] < hedef:
11              sol = orta + 1          # sağ yarıya geç
12          else:
13              sag = orta - 1          # sol yarıya geç
14
15      return -1                       # hedef bulunamadı
16
17  sayilar = [2, 5, 9, 14, 20, 38, 45]
18  aranan  = 38
19
20  sonuc = ikili_arama(sayilar, aranan)
21
22  if sonuc != -1:
23      print(f"{aranan} sayısı listede bulundu. İndeks: {sonuc}")
24  else:
25      print(f"{aranan} sayısı listede bulunamadı.")
```

EKRAN ÇIKTISI

```
38 sayısı listede bulundu. İndeks: 5
```

SIRA SİZDE

Verilen programın çıktısını bilgisayarda kodlamadan tahmin ediniz. Ekran çıktısını yazarak gerekçeleriyle açıklayınız.

```
1   def ikili_arama(liste, hedef):
2       sol, sag = 0, len(liste) - 1
3       while sol <= sag:
4           orta = (sol + sag) // 2
5           if liste[orta] == hedef:
6               return orta
7           elif liste[orta] < hedef:
8               sol = orta + 1
9           else:
10              sag = orta - 1
11      return -1
12
13  veri = [1, 4, 8, 12, 19, 25, 33, 47]
14  print(ikili_arama(veri, 19))
15  print(ikili_arama(veri, 10))
```

EKRAN ÇIKTISI (Tahmin)	GEREKÇENİZ

Doğrusal arama ve ikili arama algoritmaları kullanılacağı duruma göre seçilmelidir. Tablo 8.6'da iki arama algoritmasının özellikleri karşılaştırılmıştır.

Tablo 8.6: Doğrusal Arama ve İkili Arama Özelliklerinin Karşılaştırılması

Özellik	Doğrusal Arama	İkili Arama
Ön koşul	Sırasız listede çalışır.	Liste sıralı olmak zorundadır.
Çalışma biçimi	Baştan sona tek tek kontrol edilir.	Her adımda alanı yarıya bölünür.
En fazla adım (n elemanlı liste)	n adım	$\log_2(n)$ adım
10 elemanlı liste	En fazla 10 adım	En fazla 4 adım
1.000 elemanlı liste	En fazla 1.000 adım	En fazla 10 adım
1.000.000 elemanlı liste	En fazla 1.000.000 adım	En fazla 20 adım
Kullanım kolaylığı	Uygulaması basittir.	Önce sıralama gerektirir.
Ne zaman tercih edilir?	Küçük listeler, sırasız veri	Büyük listeler, sıralı veri

1. UYGULAMA

Arama Algoritmalarını Karşılaştırıyorum

Süre: 40 dakika

Amaç: Arama algoritmalarını aynı programda kullanarak karşılaştırmak

Yönerge: Yazılan adımları takip ederek uygulamayı tamamlayınız.

1. Adım: 10 elemanlı iki not listesi tanımlayınız. notlar listesindeki değerler sırasız, notlar1 listesindeki değerlerse küçükten büyüğe sıralı olmalıdır. Listelerin içine en az bir not değerini (örneğin 75) iki kez tekrar edecek şekilde ekleyiniz. Tanımladığınız listeye ait değerleri yazınız.

notlar = [ ]

notlar1 = [ ]

2. Adım: Bu listeyi kullanan bir program yazınız. Program iki fonksiyon içermelidir.

a) dogrusal_arama(liste, hedef)

b) ikili_arama(liste, hedef)

3. Adım: Kullanıcıdan bir not değeri girmesini isteyiniz. Doğrusal arama fonksiyonunda notlar listesini ve ikili arama fonksiyonunda ise notlar1 listesini kullanınız.

Sonuçları aşağıdaki formatta ekrana yazdırınız.

Aranan not: 78

Doğrusal Arama - İndeks: 4

İkili Arama - İndeks: 6

4. Adım: ikili_arama() metoduna sırasız notlar1 listesini göndererek listede bulunan farklı değerlerle metodu test ediniz. Veriler her defasında bulundu mu?

5. Adım: Programı listede bulunmayan değerlerle test ediniz.

6. Adım: 1. adımda listeye bilerek eklediğiniz **tekrar eden not değerini** aratınız. Doğrusal arama ve ikili aramanın ekrana yazdırdığı indeks numaralarını karşılaştırınız. Algoritmalar aynı indeksi mi buldu? Doğrusal arama ve ikili aramanın çalışma mantığını düşünerek bu durumun nedenini açıklayınız.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

SIRA SİZDE

Bir akıllı telefon rehberinde 5.000 kişinin kayıtlı olduğunu düşününüz. Aşağıda verilen soruları cevaplandırınız ve sınıfta paylaşınız.

1. Telefon rehberi alfabetik sıraya göre düzenlenmiş ve bir kişiyi aramak istiyorsanız hangi arama yöntemini tercih ederdiniz? Neden?

2. Rehber sıralı değilse ikili arama yöntemini kullanabilir misiniz? Bu durumda ne yapardınız?

3. Günlük hayatta farkında olmadan hangi durumlarda ikili aramaya benzer yöntemler kullanılmaktadır?

8.2. SIRALAMA ALGORİTMALARI

Bir veri kümesindeki elemanları belirli bir düzene (küçükten büyüğe veya büyükten küçüğe) göre yeniden düzenleyen adım adım uygulanan talimatlara **sıralama algoritması** (sorting algorithm) denir. Sıralama işlemi sadece verileri düzenli tutmak için değil ikili arama gibi verimli algoritmaların çalışabilmesi için de zorunludur.

Öğretmenin sınav kâğıtlarını numara sırasına göre dizmesi veya müzik çalar uygulamasının şarkıları alfabetik sıraya koyması birer sıralama işlemidir. Sıralama algoritmalarında kullanılan temel kavramlar Tablo 8.7'de listelenmiştir.

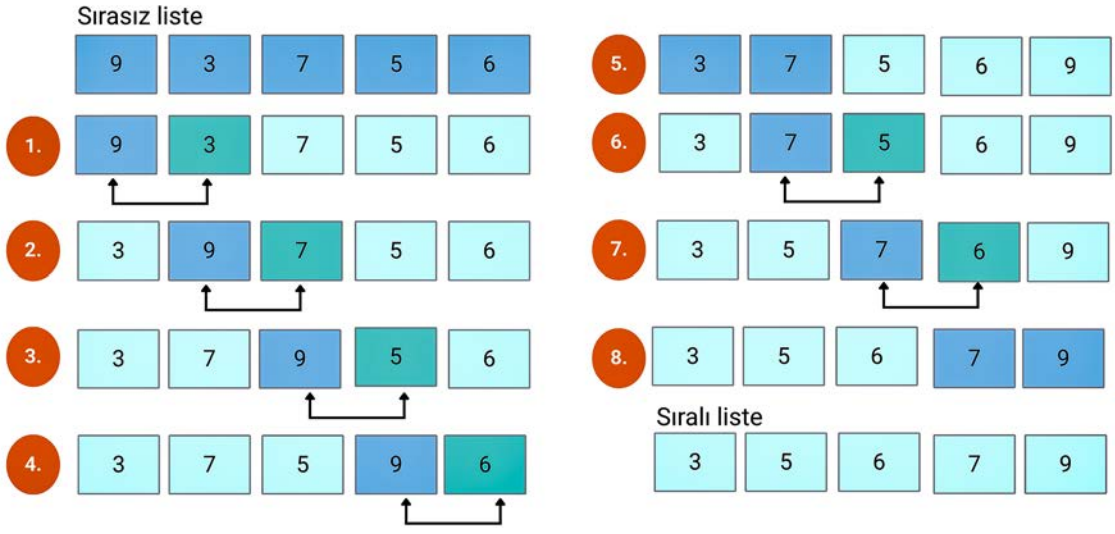
Tablo 8.7: Sıralama Algoritması Kavramları ve Açıklamaları

Kavram	Açıklama
Takas (Swap)	İki elemanın yerini değiştirme işlemi
Geçiş (Pass)	Tüm liste üzerinde bir tur yapma işlemi
Sıralı Bölüm	Listenin sıralamaya alınmış kısmı
Sırasız Bölüm	Listenin henüz sıralanmamış kısmı
Karşılaştırma Sayısı	Algoritmanın kaç adımda tamamlandığını gösteren temel ölçüt

Kabarcık Sıralama

Yan yana duran iki elemanı karşılaştırarak uygun olmayan sıradakini takas (swap) eden algoritmaya **kabarcık sıralama** (bubble sort) denir. Her geçişin (pass) sonunda en büyük eleman listenin sonuna kabarcık gibi yerleşir. Bu işlem, liste tamamen sıralanana kadar tekrar edilir.

Rastgele sıraya dizilmiş rakamlar küçükten büyüğe sıralanırken yanındakinden büyük sayı sağa, küçük olan sayı sola geçer. Bu işlem tüm rakamlar sıralanana kadar sürer (Görsel 8.2).



Görsel 8.2: Kabarcık sıralama algoritması

Tablo 8.8'de kabarcık sıralama algoritmasının çalışma mantığı, Tablo 8.9'da izleme aşamaları verilmiştir.

Tablo 8.8: Kabarcık Sıralama Algoritması Çalışma Mantığı

Adım	İşlem	Açıklama
1.	i = 0'dan başla	Dış döngü: n-1 geçiş yapılacak.
2.	j = 0 ile n-i-2 arası	İç döngü: Her geçişte bir eleman azalır.
3.	liste[j] > liste[j+1]?	Komşu çift karşılaştırılır.
4.a)	Evet ⟶ takas et	İki eleman yer değiştirir.
4.b)	Hayır ⟶ devam et	Takas gerekmez, j artar.
5.	Tüm geçişler bitti mi?	Evet ⟶ sıralama tamamlandı.

Tablo 8.9: Adım Adım İzleme (liste = [5, 3, 8, 1, 4] (Geçiş 1))

Karşılaştırma	Liste Durumu	İşlem	Takas?
5 ve 3	[5, 3, 8, 1, 4]	5 > 3	✓ Evet
5 ve 8	[3, 5, 8, 1, 4]	5 < 8	Hayır
8 ve 1	[3, 5, 8, 1, 4]	8 > 1	✓ Evet
8 ve 4	[3, 5, 1, 8, 4]	8 > 4	✓ Evet
Geçiş 1 sonu	[3, 5, 1, 4, 8]	En büyük (8) sona yerleşti	–

BİLGİ KUTUSU

Her geçişin sonunda o geçişin en büyük elemanı, listenin sonuna yerleşir ve bir sonraki geçişe dahil edilmez. Beş elemanlı bir liste için en fazla dört geçiş gerekir. Her geçişte bir eleman yerli yerine oturur.

3. ÖRNEK UYGULAMA

Amaç: Kabarcık sıralama algoritmasını uygulayan program yazmak

```
1   def kabarcik_siralama(liste):
2       n = len(liste)
3       for i in range(n - 1):
4           for j in range(n - i - 1):
5               if liste[j] > liste[j + 1]:
6                   # takas işlemi
7                   liste[j], liste[j + 1] = liste[j + 1], liste[j]
8           print(f"Geçiş {i + 1}: {liste}")
9       return liste
10
11  sayilar = [5, 3, 8, 1, 4]
12  print("Başlangıç: ", sayilar)
13  sonuc = kabarcik_siralama(sayilar)
14  print("Sıralanmış:", sonuc)
```

EKRAN ÇIKTISI

Başlangıç: [5, 3, 8, 1, 4]
Geçiş 1: [3, 5, 1, 4, 8]
Geçiş 2: [3, 1, 4, 5, 8]
Geçiş 3: [1, 3, 4, 5, 8]
Geçiş 4: [1, 3, 4, 5, 8]
Sıralanmış: [1, 3, 4, 5, 8]

SIRA SİZDE

3. Örnek Uygulama'da yazdığınız programda listeyi büyükten küçüğe sıralamanız gerekseydi hangi numaralı satırlarda hangi değişiklikleri yapmanız gerekirdi? Gerekçeleri ile açıklayınız.

.....

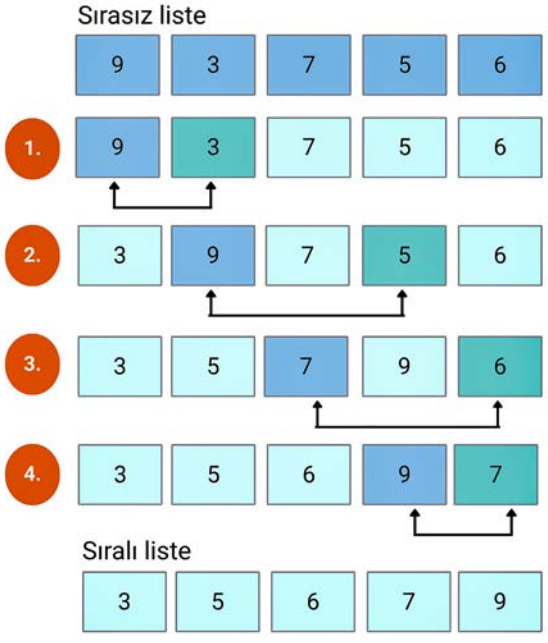
.....

.....

Seçmeli Sıralama

Sırasız bölümden her seferinde en küçük elemanı seçip sıralı bölümün sonuna ekleyen algoritmaya **seçmeli sıralama** (selection sort) denir. Her geçişte listede bir eleman yerli yerine otururken sırasız bölüm bir azalır.

Rastgele sıraya dizilmiş rakamlar küçükten büyüğe sıralanırken tüm sayılara bakılarak en küçük sayı seçilir ve en başa yerleştirilir. Sonra kalan sayılardan en küçük olanı seçilerek sıraya yerleştirilir. Bu işlem tüm sayılar sıralanana kadar sürer (Görsel 8.3).



Görsel 8.3: Seçmeli sıralama algoritması

Tablo 8.10'da seçmeli sıralama algoritmasının çalışma mantığı, Tablo 8.11'de adım adım izleme aşamaları verilmiştir.

Tablo 8.10: Seçmeli Sıralama Algoritması Çalışma Mantığı

Adım	İşlem	Açıklama
1.	i = 0'dan başla	Sırasız bölümün başlangıç indeksi
2.	min_indeks = i	Şimdilik en küçük eleman bu konumda
3.	j = i+1'den sona tara	Sırasız bölümde daha küçük var mı?
4.	liste[j] < liste[min_indeks]?	Evet ⟶ min_indeks = j
5.	Takas: liste[i] ⟷ liste[min_indeks]	En küçük eleman sıralı bölümün sonuna gider.

Tablo 8.11: Adım Adım İzleme (liste = [5, 3, 8, 1, 4])

Geçiş	Sırasız Bölüm	Bulunan En Küçük	Liste (geçiş sonu)
1.	[5, 3, 8, 1, 4]	1 (indeks 3)	[1 | 3, 8, 5, 4]
2.	[3, 8, 5, 4]	3 (indeks 1)	[1, 3 | 8, 5, 4]
3.	[8, 5, 4]	4 (indeks 4)	[1, 3, 4 | 5, 8]
4.	[5, 8]	5 (indeks 3)	[1, 3, 4, 5 | 8]
5. (son)	[8]	8 (tek eleman)	[1, 3, 4, 5, 8]

BİLGİ KUTUSU

Sıralı bölüm "|" (pipe) karakteriyle gösterilmiştir. Her geçişte sıralı bölümde bir eleman artar. Seçmeli sıralama her durumda **aynı sayıda** karşılaştırma yapar. Beş elemanlı listede dört geçiş, n elemanlı listede n-1 geçiş gerekir.

4. ÖRNEK UYGULAMA

Amaç: Seçmeli sıralama algoritmasını uygulayan program yazmak

```
1   def secmeli_siralama(liste):
2       n = len(liste)
3       for i in range(n - 1):
4           min_indeks = i
5           for j in range(i + 1, n):
6               if liste[j] < liste[min_indeks]:
7                   min_indeks = j
8           # en küçük eleman sıralı bölümün sonuna taşınır
9           liste[i], liste[min_indeks] = liste[min_indeks], liste[i]
10          print(f"Geçiş {i + 1}: {liste}")
11      return liste
12
13  sayilar = [5, 3, 8, 1, 4]
14  print("Başlangıç: ", sayilar)
15  sonuc = secmeli_siralama(sayilar)
16  print("Sıralanmış:", sonuc)
```

EKRAN ÇIKTISI

Başlangıç: [5, 3, 8, 1, 4]
Geçiş 1: [1, 3, 8, 5, 4]
Geçiş 2: [1, 3, 8, 5, 4]
Geçiş 3: [1, 3, 4, 5, 8]
Geçiş 4: [1, 3, 4, 5, 8]
Sıralanmış: [1, 3, 4, 5, 8]

SIRA SİZDE

Verilen programın çıktısını bilgisayarda **kodlamadan** tahmin ediniz. Ekran çıktısını yazarak gerekçeleriyle açıklayınız.

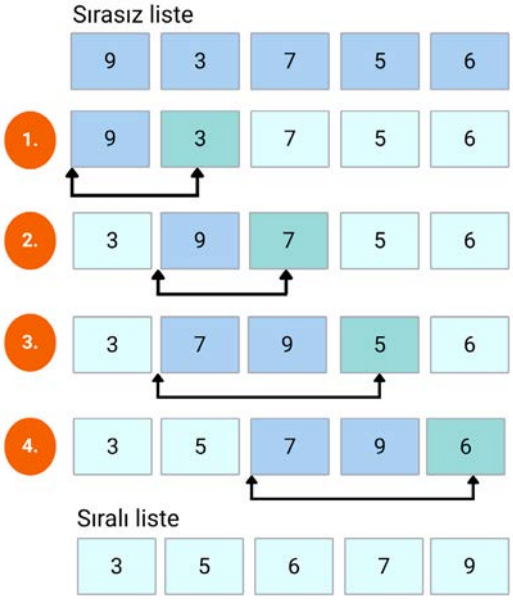
```
1   def secmeli_siralama(liste):
2       n = len(liste)
3       for i in range(n - 1):
4           min_indeks = i
5           for j in range(i + 1, n):
6               if liste[j] < liste[min_indeks]:
7                   min_indeks = j
8           liste[i], liste[min_indeks] = liste[min_indeks], liste[i]
9       return liste
10
11  veri = [10, 1, 7, 3, 9, 2]
12  print(secmeli_siralama(veri))
```

EKRAN ÇIKTISI

GEREKÇESİ

Her adımda sırasız bölümden bir eleman alarak sıralı bölümdeki doğru konumuna yerleştiren algoritmaya, **eklemeli sıralama** (insertion sort) denir. Sol taraf her zaman sıralı tutulur; yeni eleman bu sıralı bölümde uygun yere yerleştirilir.

Rastgele sıraya dizilmiş rakamlar küçükten büyüğe sıralanırken sırası gelen sayı solda doğru yerine yerleştirilir. Bu işlem tüm sayılar sıralanana kadar devam eder (Görsel 8.4).



Görsel 8.4: Eklemeli sıralama algoritması

Tablo 8.12'de eklemeli sıralama algoritmasının çalışma mantığı, Tablo 8.13'te izleme aşamaları verilmiştir.

Tablo 8.12: Eklemeli Sıralama Çalışma Mantığı

Adım	İşlem	Açıklama
1.	i = 1'den başla	0. eleman zaten tek başına sıralıdır.
2.	anahtar = liste[i]	Sırasız bölümden alınan yeni eleman
3.	j = i-1, geriye doğru tara	Sıralı bölümde yer arıyoruz.
4.	liste[j] > anahtar?	Evet ⟶ liste[j+1] = liste[j], j -= 1
5.	liste[j+1] = anahtar	Doğru konum bulundu, anahtarı yerleştir.

Tablo 8.13: Adım Adım İzleme (liste = [5, 3, 8, 1, 4])

Adım	Anahtar	Sıralı Bölüm (önce)	Liste (sonra)
i=1	3	[5]	[3, 5 | 8, 1, 4]
i=2	8	[3, 5]	[3, 5, 8 | 1, 4]
i=3	1	[3, 5, 8]	[1, 3, 5, 8 | 4]
i=4	4	[1, 3, 5, 8]	[1, 3, 4, 5, 8]

BİLGİ KUTUSU

Sıralı bölüm "|" karakteriyle gösterilmiştir. Her adımda bir eleman sıralı bölüme katılır. Anahtar eleman sıralı bölümdeki **doğru konumuna girerken** büyük elemanlar bir sağa kayar. Eklemeli sıralama, verinin zaten büyük ölçüde sıralı olduğu ve araya yeni eleman eklenmesi gereken durumlarda çok verimlidir.

5. ÖRNEK UYGULAMA

Amaç: Eklemeli sıralama algoritmasını uygulayan program yazmak

```
1   def eklemeli_siralama(liste):
2       for i in range(1, len(liste)):
3           anahtar = liste[i]
4           j = i - 1
5           # anahtar için doğru konum bulunana kadar elemanları sağa kaydır
6           while j >= 0 and liste[j] > anahtar:
7               liste[j + 1] = liste[j]
8               j -= 1
9           liste[j + 1] = anahtar
10          print(f"Adım {i}: {liste}")
11      return liste
12  sayilar = [5, 3, 8, 1, 4]
13  print("Başlangıç: ", sayilar)
14  sonuc = eklemeli_siralama(sayilar)
15  print("Sıralanmış:", sonuc)
```

EKRAN ÇIKTISI

```
Başlangıç:  [5, 3, 8, 1, 4]
Adım 1: [3, 5, 8, 1, 4]
Adım 2: [3, 5, 8, 1, 4]
Adım 3: [1, 3, 5, 8, 4]
Adım 4: [1, 3, 4, 5, 8]
Sıralanmış: [1, 3, 4, 5, 8]
```

SIRA SİZDE

5. Örnek Uygulama'da Adım 1 ile Adım 2 neden aynı çıktıyı üretir? Gerekçesini yazınız.

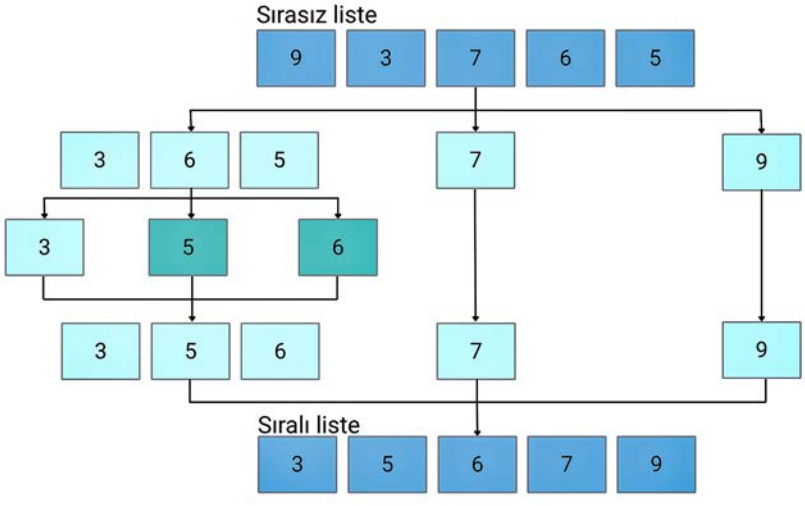
.....

.....

Hızlı Sıralama

Parçala ve yönet stratejisine dayanan, veri kümesini bir **dayanak** (pivot) eleman seçerek ikiye bölen ve bu işlemi **özyinelemeli** olarak gerçekleştiren algoritmanın adına **hızlı sıralama** (quick sort) adı verilir. Seçilen dayanak elemanından küçük olan veriler pivotun soluna, büyük olanlar ise sağına yerleştirilir. Bu işlem sonucunda pivot eleman, sıralanmış dizideki kesin yerine yerleşmiş olur.

Rastgele sıraya dizilmiş rakamlar küçükten büyüğe sıralanırken rastgele bir sayı seçilir (pivot). Bu sayıdan küçük olanlar sol tarafa, büyük olanlar ise sağ tarafa geçer. Seçilen sayı artık sıradaki tam ve doğru yerindedir. Ardından solda ve sağda oluşan iki yeni grup için de aynı yöntem uygulanır. Tüm sayılar doğru sıraya gelene kadar işlem devam eder (Görsel 8.5).



Görsel 8.5: Hızlı sıralama algoritması

Tablo 8.15'te hızlı sıralama algoritmasının çalışma mantığı, Tablo 8.16'da izleme aşamaları verilmiştir.

Tablo 8.15: Hızlı Sıralama Algoritması Çalışma Mantığı

Adım	İşlem	Açıklama
1.	pivot = liste[-1]	Listenin son elemanı pivot olarak seçilir.
2.	küçükler = [], büyükler = []	Pivot dışındaki elemanlar iki gruba ayrılır.
3.	eleman < pivot?	Evet ⟶ küçükler listesine ekle
4.	eleman >= pivot?	Evet ⟶ büyükler listesine ekle
5.	Özyineleme (recursive call)	küçükler ve büyükler için aynı işlem tekrar uygulanır.
6.	Sonuç = küçükler + [pivot] + büyükler	Parçalar birleştirilir; liste sıralanmış olur.

Tablo 8.16: Adım Adım İzleme (liste = [5, 3, 8, 1, 4])

Çağrı	Liste	Pivot	Küçükler	Büyükler	Sonuç
1. çağrı	[5, 3, 8, 1, 4]	4	[3, 1]	[5, 8]	devam
2. çağrı	[3, 1]	1	[]	[3]	devam
2. çağrı	[5, 8]	8	[5]	[]	devam
Birleşim	[] + [1] + [3]	—	[1, 3]	—	[1,3]
Birleşim	[5] + [8] + []	—	[5, 8]	—	[5,8]
Son	[1,3] + [4] + [5,8]	—	—	—	✓ Bitti

BİLGİ KUTUSU

Ortalama durumda hızlı sıralama $n \times \log_2(n)$ adımda tamamlanır. Özellikle büyük veri kümelerinde çok verimlidir.

6. ÖRNEK UYGULAMA

Amaç: Hızlı sıralama algoritmasını uygulayan program yazmak

```
1  def hizli_siralama(liste):
2      if len(liste) <= 1:
3          return liste          # Tek eleman veya boş liste zaten sıralıdır
4      pivot    = liste[-1]      # Son eleman pivot olarak seçilir
5      kucukler = [x for x in liste[:-1] if x <  pivot]
6      buyukler = [x for x in liste[:-1] if x >= pivot]
7      print(f"Pivot: {pivot}  |  Küçükler: {kucukler}  |  Büyükler: {buyukler}")
8      return hizli_siralama(kucukler) + [pivot] + hizli_siralama(buyukler)
9
10 sayilar = [5, 3, 8, 1, 4]
11 print("Başlangıç :", sayilar)
12 sonuc = hizli_siralama(sayilar)
13 print("Sıralanmış:", sonuc)
```

EKRAN ÇIKTISI

```
Başlangıç : [5, 3, 8, 1, 4]
Pivot: 4  |  Küçükler: [3, 1]  |  Büyükler: [5, 8]
Pivot: 1  |  Küçükler: []  |  Büyükler: [3]
Pivot: 8  |  Küçükler: [5]  |  Büyükler: []
Sıralanmış: [1, 3, 4, 5, 8]
```

2. UYGULAMA

Sıralama Algoritmalarını Karşılaştırıyorum

Süre: 40+40 dakika

Amaç: Sıralama algoritmalarını aynı programda kullanarak karşılaştırmak

Yönerge: Yazılan adımları takip ederek uygulamayı tamamlayınız.

1. Adım: Verilen listede **mükemmel sayılar** (mükemmel sayı kendisi hariç tüm pozitif bölenlerinin toplamına eşit olan sayıdır.)**, asal sayılar** ve normal sayılardan oluşan karışık bir liste bulunmaktadır. Bu sayılardan hangilerinin mükemmel sayı, hangilerinin asal sayı olduğunu yazınız.

gizemli_sayilar = [496, 13, 28, 7, 6, 31, 8128, 2, 17, 8]

Mükemmel Sayılar: [ ]

Asal Sayılar: [ ]

Sıralanacak karışık listeyi programda tanımlayınız.

2. Adım: Bu listeyi kullanan bir Python programı yazınız. Program dört fonksiyon içermelidir.

- kabarcik_siralama(liste)
- secmeli_siralama(liste)
- eklemeli_siralama(liste)
- hizli_siralama(liste)

3. Adım: Her fonksiyonun içine algoritmanın temel adımlarını sayacak bir sayaç değişkeni (islem_sayisi = 0) ekleyiniz. Sayaç, programın yaptığı her bir **karşılaştırma (if x > y)** ve her bir **yer değiştirme/kaydırma** işleminde **1** artırılmalıdır. Hızlı sıralama fonksiyonunda her özyinelemeli çağrının da bu toplam işlem sayısına dahil edildiğinden emin olunuz.

4. Adım: Sonuçları verilen formatta ekrana yazdırınız. (A, B, C ve Ç değerleri program çalıştırılınca görülecektir.)

EKRAN ÇIKTISI

```
ALGORİTMA  |   İŞLEM SAYISI
-----------------------------------
Kabarcık Sıralama   | A
Seçmeli Sıralama    | B
Eklemeli Sıralama   | C
Hızlı Sıralama      | Ç
```

5. Adım: Yazdığınız programa göre aşağıdaki soruları cevaplandırınız.

a) Hangi sıralama işlemi en az adımda tamamlandı? Gerekçeleri ile açıklayınız.

.....

b) Daha fazla adımda tamamlanan diğer üç sıralama algoritmasının adım sayılarına göre algoritma hızlarını değerlendiriniz.

.....

c) gizemli_sayilar listesini genişleterek karışık sırada 20 sayıya çıkartınız. Listede 10 sayı olması ile 20 sayı olması arasındaki adım farkını karşılaştıran tabloyu doldurunuz.

Algoritma	10 Sayı Sıralama	20 Sayı Sıralama
Kabarcık Sıralama		
Seçmeli Sıralama		
Eklemeli Sıralama		
Hızlı Sıralama		

ç) Listedeki sayı adedi arttıkça sıralama adımların **artış oranını** dikkate alınız. Daha büyük bir listeyi sıralamanız gerekirse hangi sıralamayı kullanırsınız? Gerekçeleri ile açıklayınız.

.....

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

7. ÖRNEK UYGULAMA

Amaç: Kabarcık sıralama algoritmasıyla metin verilerini sıralamak

```
def alfabetik_sirala(kelime_listesi):
    n = len(kelime_listesi)
    for i in range(n - 1):
        for j in range(n - i - 1):
            # İki metni (string) alfabetik olarak karşılaştır
            if kelime_listesi[j] > kelime_listesi[j + 1]:
                # Alfabetik sırada sonra geleni sağa kaydır (Takas)
                kelime_listesi[j], kelime_listesi[j + 1] = kelime_listesi[j + 1], kelime_listesi[j]
    return kelime_listesi
isimler = ["Zeynep", "Ali", "Hasan", "Cemil", "Ayşe"]
print("Başlangıç:", isimler)
sonuc = alfabetik_sirala(isimler)
print("Sıralanmış:", sonuc)
```

EKRAN ÇIKTISI

Başlangıç: [‘Zeynep’, ‘Ali’, ‘Hasan’, ‘Cemil’, ‘Ayşe’]

Sıralanmış: [‘Ali’, ‘Ayşe’, ‘Cemil’, ‘Hasan’, ‘Zeynep’]

8.1. PROJE	
Konu	Akıllı Kütüphane Sistemine Arama ve Sıralama Özellikleri Ekliyorum
Kapsanan Konular	Doğrusal arama, ikili arama, kabarcık sıralama, seçmeli sıralama, eklemeli sıralama, hızlı sıralama, kullanıcı menüsü
Süre	40+40+40 dakika
Amaç	Arama ve sıralama algoritmalarını kapsayıcı bir projede kullanmak
Çalışma Biçimi	Bireysel

Proje Senaryosu

Nesne yönelimli programlama (OOP) öğrenme biriminde okul kütüphanesi için proje geliştirmiştiniz. Ancak eklediğiniz her kitap, listenin sonuna ekleniyordu ve bir kitabı bulmak istediğinizde programınız yavaş çalışarak aramayı verimsiz hâle getiriyordu. Bu sorunu çözmek için daha etkili yöntemlere ihtiyaç duyuldu. Bunun için projede yazılacak program; kitap listesini dört farklı sıralama algoritmasından kullanıcının seçtiği yöntemle sıralayabilmelidir. Sıralanan listede hem doğrusal hem de ikili arama yapabilmeli ve kullanıcıya menü arayüzü sunarak hangi işlemi yapmak istediğini sorabilmelidir.

1. Aşama: Sıralama metodlarını kodlayınız.

Kapsanan İlke: Kabarcık sıralama, seçmeli sıralama, eklemeli sıralama, hızlı sıralama

1. Görev

a) En az 10 kitabı program içerisinde tanımlayınız.
b) Kitap adına göre sıralama metodlarını yazınız.
c) Sıralama metodları listeyi değiştirmemelidir. Her sıralama işlemi öncesi liste_kopya=liste.copy() komutu ile listenin bir kopyasını almayı unutmayınız.

2. Aşama: Arama metodlarını kodlayınız.

Kapsanan İlke: Doğrusal arama, ikili arama

2. Görev

a) Kitap adına göre arama metodlarını yazınız.

3. Aşama: Kullanıcı menüsü sistemini kodlayınız.

Kapsanan İlke: Kullanıcı girişi, döngüsel menü

3. Görev

a) Aşağıdaki menü yapısını oluşturunuz.

Menü Tasarımı
========== KÜTÜPHANE YÖNETİM SİSTEMİ ========== 1. Kitap Listesini Sırala a) Kabarcık Sıralama b) Seçmeli Sıralama c) Eklemeli Sıralama d) Hızlı Sıralama 2. Kitap Ara a) Doğrusal Arama b) İkili Arama (önce sıralama gerekir.) 3. Tüm Kitapları Listele 0. Çıkış Seçiminiz: _

b) Seçime göre ilgili metot çalıştırılır ve işlem bittiğinde menüye dönülür.

4. Aşama: Yazdığınız programı test ediniz.

Kapsanan İlke: Hata yönetimi, test senaryoları

4. Görev

a) Sıralama metodlarını test ederek doğru sıralamalı çıktıyı ürettiğini kontrol ediniz.
b) Türkçeye özgü karakter içeren ve içermeyen kitap adlarına göre arama yaparak gözlemleyiniz ve sebebini açıklayınız.
c) Listede olmayan kitap adlarını arayarak **kitap bulunamadı** mesajının üretildiğinden emin olunuz.
ç) İkili arama seçeneğinde önce listenin sıralı olup olmadığı kontrol edilmelidir. Liste sıralı değilse uyarı mesajı üretilerek menüye dönülmelidir.
d) Menüye hatalı giriş koruması (try-except) ekleyiniz. Kullanıcı tanımsız bir menü değeri girdiğinde program çökmemeli, uyarı verip menüye dönmelidir.

5. Aşama: Projenize yeni özellikler ekleyiniz.

Kapsanan İlke: Yaratıcı geliştirme

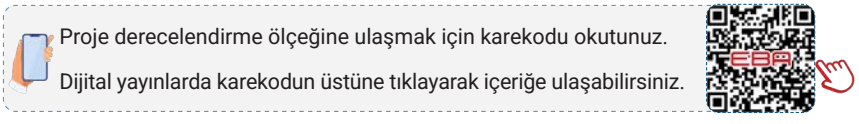
a) Verilen geliştirme önerilerinden en az ikisini seçerek projenize ekleyiniz.

I. Menüye yazar adına göre ve yıla göre sıralama seçenekleri ekleyiniz.
II. Kitap adına göre kısmi arama seçeneği ekleyiniz. Örneğin **Safa** yazılarak arandığında Safahat kitabı bulunabilmelidir.
III. Menüye **İstatistikler** seçeneği ekleyiniz. En eski kitap, en yeni kitap, ortalama yayın yılı gibi bilgileri listeleyebilmelidir.
IV. Menüye **Kitap Ekle** ve **Kitap Çıkar** seçenekleri ekleyiniz. Kullanıcıdan yeni kitap bilgilerini alarak listeye ekleme ve listeden çıkarma işlemlerini gerçekleştiriniz.

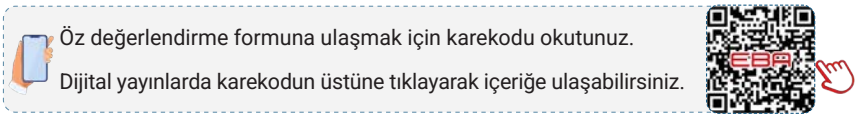
6. Aşama: Projenizin güncel kodlarını dijital deponuza gönderiniz. README.md dosyasını güncellemeyi unutmayınız.

Değerlendirme

a) Yapacağınız proje çalışması öğretmeniniz tarafından **Proje Derecelendirme Ölçeğinde** yer alan ölçütlere göre değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak derecelendirme ölçeğine ulaşabilirsiniz.



b) **Öz Değerlendirme Formu** proje geliştirme sürecinin sonunda kendinizi değerlendirmeniz amacıyla hazırlanmıştır. Cevaplarınızı tabloda ilgili kutucuğa yazınız. Verilen karekodu kullanarak öz değerlendirme formuna ulaşabilirsiniz.



ÖLÇME VE DEĞERLENDİRME

A) Aşağıdaki yapılandırılmış gridde, numaralandırılmış kutucuklarda arama ve sıralama algoritmaları ile ilgili bazı ifadeler verilmiştir. Kutucuk numaralarını kullanarak soruları cevaplayınız. Aynı kutucuk numarasını birden fazla cevap için kullanabilirsiniz.

(1) Doğrusal arama	(2) İkili arama	(3) Kabarcık sıralaması
(4) Seçmeli sıralama	(5) Eklemeli sıralama	(6) Takas
(7) Hızlı Sıralama	(8) Pivot	(9) Parçala ve Yönet

1. Bir veri topluluğu içerisinde hedeflenen değeri veya anahtarı tespit etmek amacıyla kullanılan **arama algoritmaları** hangileridir? (.....)

2. Büyük bir veri kümesini daha küçük alt kümelere ayırarak arayan **sıralama yöntemi** nedir, bu yöntemi doğrudan kullanan **sıralama algoritması** hangisidir? (.....)

3. Listedeki elemanları birbiriyle karşılaştırıp **iki elemanın yerini değiştirme** mekanizmasını kullanan algoritmalar ve kavramlar hangileridir? (.....)

4. Büyük veri setlerinde verimliliği düşük ancak küçük veri gruplarında araya ekleme veya komşu elemanları kaydırıp yer değiştirme mantığıyla çalışan **doğrusal sıralama algoritmaları** hangileridir? (.....)

5. Hızlı Sıralama algoritmasında veri kümesini kendisinden küçükler ve büyükler olarak ayıran **referans elemana** ne ad verilir? (.....)

6. Bir veri setinde arama işleminin başlayabilmesi için verilerin **mutlaka önceden sıralanmış olması** ön koşuluna sahip olan algoritma hangisidir? (.....)

7. Her adımda dizinin henüz sıralanmamış kısmını tarayıp elemanı sırasız kısmın başındaki elemanla yer değiştirerek ilerleyen **algoritma** ve bu algoritmanın kullandığı **temel takas eylemi** nedir? (.....)

8. En kötü durum senaryosunda performans kaybı yaşamamak için baştan sona tüm elemanları **tek tek kontrol eden** yöntemler hangileridir? (.....)

B) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1.

```
def dogrusal_arama(liste, hedef):
    for i in range(len(liste)):
        if liste[i] == hedef:
            return i
    return -1
dogrusal_arama([10, 30, 50, 70, 90], 70)
```

Yukarıdaki kod bloğu çalıştırıldığında kaç karşılaştırma yapılır?

A) 1
B) 2
C) 3
D) 4
E) 5

2. İkili arama algoritmasının doğru çalışabilmesi için temel ön koşul nedir?

A) Listenin elemanları tam sayı olmalıdır.
B) Liste küçükten büyüğe veya büyükten küçüğe sıralı olmalıdır.
C) Listenin eleman sayısı çift olmalıdır.
D) Aranan eleman listenin ilk yarısında bulunmalıdır.
E) Liste en az 10 eleman içermelidir.

3.

```
liste = [8, 3, 6, 1, 9]
n = len(liste)
for i in range(n - 1):
    for j in range(n - i - 1):
        if liste[j] > liste[j + 1]:
            liste[j], liste[j + 1] = liste[j + 1], liste[j]
print(liste)
```

Yukarıdaki kod bloğu çalıştırıldığında hangi çıktı elde edilir?

A) [8, 3, 6, 1, 9]
B) [1, 3, 6, 8, 9]
C) [9, 8, 6, 3, 1]
D) [1, 3, 8, 6, 9]
E) Hata mesajı üretir.

4. 1.000.000 elemanlı sıralı bir listede ikili arama yöntemiyle arama yapıldığında en fazla kaç adım gerekir?

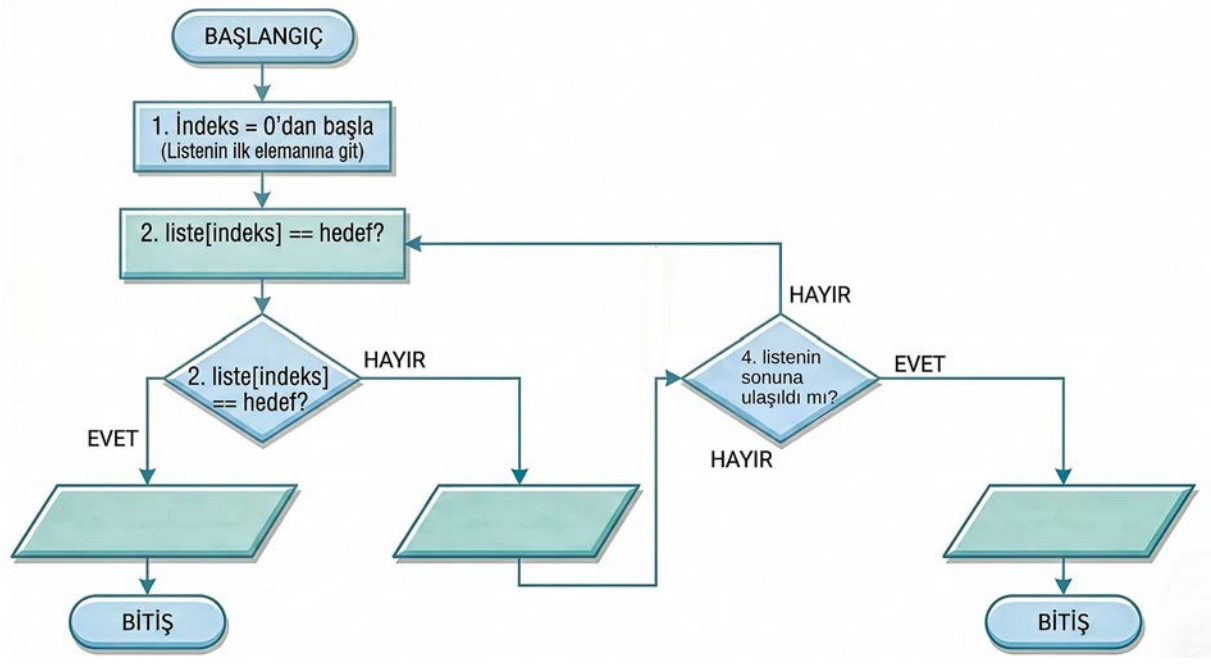
A) 1.000.000
B) 500.000
C) 1000
D) 20
E) 10

5. Aşağıdakilerden hangisi seçmeli sıralama algoritmasının doğru tanımıdır?

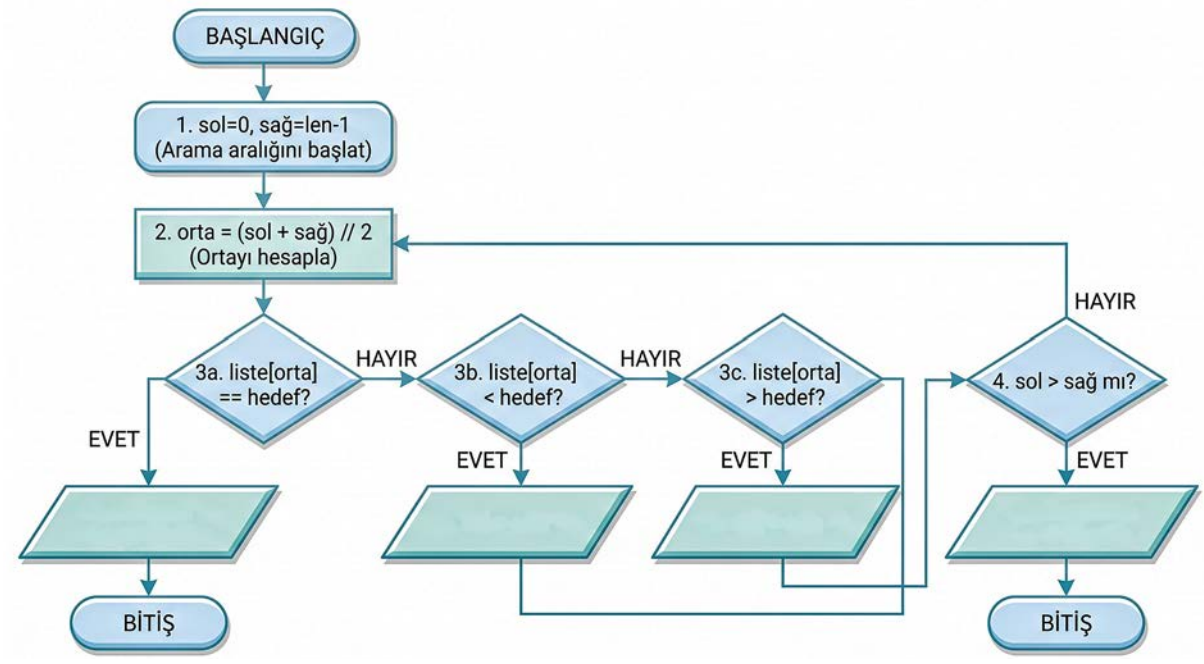
A) Komşu elemanları karşılaştırarak yanlış sıradakileri takas eder.
B) Her adımda sırasız bölümden en küçük elemanı seçip sıralı bölümün sonuna ekler.
C) Her yeni elemanı sıralı bölümdeki doğru konumuna ekler.
D) Listeyi ikiye bölerek her yarıyı ayrı ayrı sıralar.
E) Listeyi sıralarken yalnızca tek bir geçiş yapar.

C) Aşağıda verilen akış diyagramlarında boş bırakılan yerleri doldurunuz.

a) Doğrusal arama akış diyagramı



b) İkili arama akış diyagramı



Ç) 1-3. soruları aşağıdaki metne göre cevaplayınız.

Kerem kendisine verilen [42, 15, 8, 23, 54, 3, 11] listesi üzerinde bazı işlemler yapmak istiyor. Uyguladığı adımlar şunlardır:

1. Adım: Listede 11 değerini bulmak için doğrudan ikili arama algoritmasını başlatıyor ve listenin tam ortasındaki 23 değerine bakıyor. 11, 23'ten küçük olduğu için sağ tarafı eleyip sol tarafta aramaya devam ediyor.

2. Adım: Listeyi küçükten büyüğe sıralamak için seçmeli sıralama kullanmaya karar veriyor. Algoritmanın ilk turunda listenin en sonundaki elemanı 11 seçiyor ve bunu listenin en başındaki elemanla (42) takas ediyor.

3. Adım: Sıralama işlemini hızlandırmak için kabarcık sıralamaya geçiyor. Her adımda komşu iki elemanı karşılaştırıyor ancak sadece birinci eleman ikinci elemandan küçükse yer değiştirme işlemi yapıyor.

1. **Kerem'in uygulama esnasında yaptığı hatalı adımlar hangileridir? Bu adımlardaki mantık hatalarını açıklayınız.**

2. **Kerem'in yaptığı bu hataların düzeltilmesi ve algoritmaların doğru sonuç vermesi için her bir adımda ne tür teknik değişiklikler yapılmalıdır?**

3. **Projelerinizde arama ve sıralama algoritmalarını seçerken bu tür hatalara yol açmamak için kodlamaya başlamadan önce nasıl bir strateji izlemelisiniz?**

D) Bu öğrenme biriminde edindiklerinizi değerlendirmek için aşağıdaki çıkış kartını doldurunuz.

Üç Yaz	Arama ve Sıralama Algoritmaları öğrenme biriminde öğrendiğiniz üç bilgiyi yazınız..
1	
2	
3	
İki Sor	Sıralama algoritmaları ile ilgili merak ettiğiniz iki soruyu yazınız.
1	
2	
Bir Paylaş	Bu öğrenme biriminde geliştirdiğiniz uygulamalardan birini sınıf arkadaşlarınızla paylaşmak isteseniz hangisini seçerdiniz? Neden?



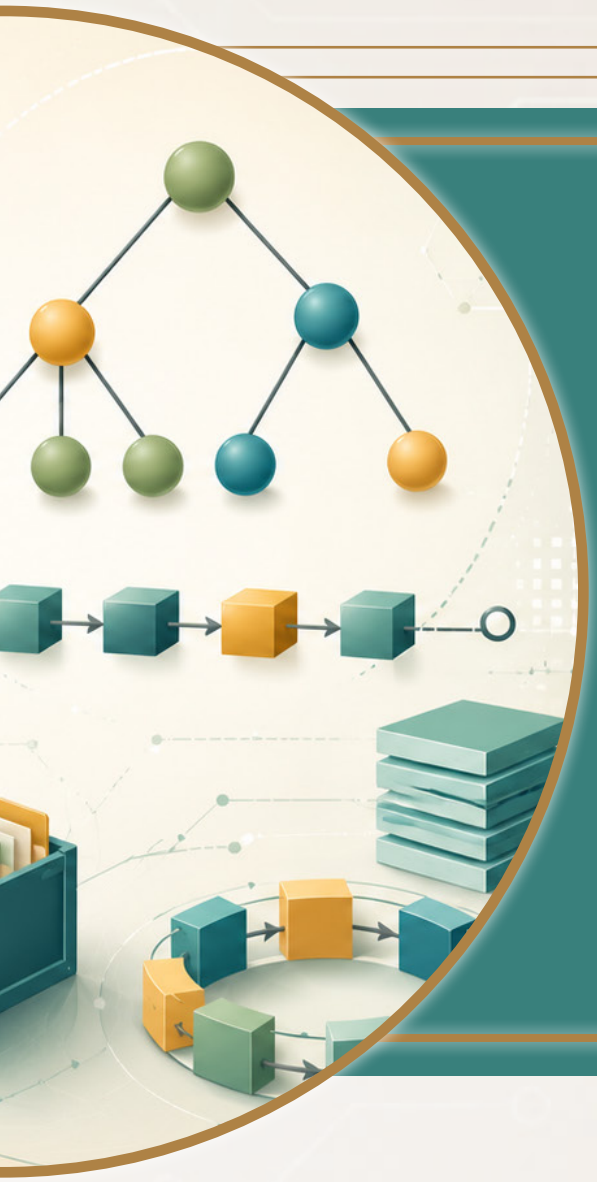
KONULAR

9.1. VERİ YAPILARI

9.2. DOSYA İŞLEMLERİ

ANAHTAR KAVRAMLAR

- ağaç
- AVL ağacı
- bağlı liste
- doğrusal yapı
- dosya işlemleri
- düğüm
- geçici bellek
- hiyerarşik yapı
- ikili arama ağacı
- kalıcı bellek
- kuyruk
- veri yapısı
- yığın



9. ÖĞRENME BİRİMİ

TEMEL VERİ YAPILARI

BEKLENTİLER

Bu öğrenme biriminde sizden

- veri yapılarını kullanmanız
- program içerisinde dosya işlemelerini uygulamanız

beklenmektedir.

HAZIRLIK ÇALIŞMALARI

1. Aşağıda yazılım geliştiricinin günlüğünden bir bölüm verilmiştir.

Binlerce şarkı arasından istediğim şarkıyı ikili arama algoritmasıyla anında bulabiliyorum. Bu da yazdığım arama ve sıralama algoritmalarının sorunsuz çalıştığını gösteriyor. Şarkıları çıkış yılı veya sanatçı adına göre de kolayca listeleyebiliyorum ancak üçüncü ile dördüncü şarkı arasına yeni bir şarkı eklediğimde ve yüzüncü şarkıyı listeden çıkardığımda bellekteki verilerin kaydırılması çok zaman alıyor. Bu durum programın anlık olarak donmasına yol açıyor. Buna bir çözüm bulamazsam programım kullanışsız hâle gelecek.

Verilen metne göre aşağıdaki soruları cevaplayınız.

a) Yazılım geliştirici şarkıları nasıl arıyor ve nasıl sıralıyor?

b) Daha önce öğrendiğiniz arama ve sıralama algoritmalarını kullanarak, programın anlık donması nasıl engellenebilir?

2. Sizce internet arama motorları, milyarlarca sayfa içerisinden anahtar kelimeyi çok kısa sürede nasıl bulabiliyor? Daha önce öğrendiğiniz arama algoritmaları ile büyük veri (big data) neden sorgulanamaz?

BAŞLARKEN

Bir müzik çalarda **Sonraki** düğmesine basıldığında sıradaki şarkıya geçilir. Tarayıcıdaki **Geri** düğmesi kullanıcıyı bir önceki sayfaya döndürür. Yazıcıya aynı anda birden fazla belge gönderildiğinde belgeler sıraya girer ve ilk gönderilen belge önce yazdırılır. Telefon rehberinde bir isim aranırken tüm isimler alfabetik sıraya göre aranır. Bu deneyimlerin hepsinin arkasında ortak bir mantık vardır. Yapılmak istenen işlemleri bellekte en hızlı ve en verimli biçimde gerçekleştirebilmek için veri yapıları kullanılır.

9.1. VERİ YAPILARI

Verileri bilgisayar belleğinde **düzenli** ve **verimli** biçimde saklayarak üzerinde işlem yapılmasını sağlayan organizasyon şemasına **veri yapısı** (data structure) denir. Doğru veri yapısını seçmek bir programın hızını, bellek kullanımını ve genel performansını doğrudan etkiler. Bir programda veriler; sıralı erişim, hızlı arama, öncelik sırası belirleme, hiyerarşik ilişkileri gösterme gibi amaçlarla saklanır. Her amaç için en uygun veri yapısını seçmek, yazılımın kalitesini belirleyen temel kararlardan biridir.

Veri yapılarında sık kullanılan temel kavramlar şunlardır:

Düğüm (Node): Veri yapısındaki her bir veri parçasıdır. Veriyi ve bağlantıyı (pointer/next) barındırır.

İşaretçi (Pointer): Bir düğümün bellekteki başka bir düğümü göstermesi için kullanılan referanstır.

Baş (Head): Bir düğümün bellekteki başka bir düğümü göstermesi için kullanılan referanstır.

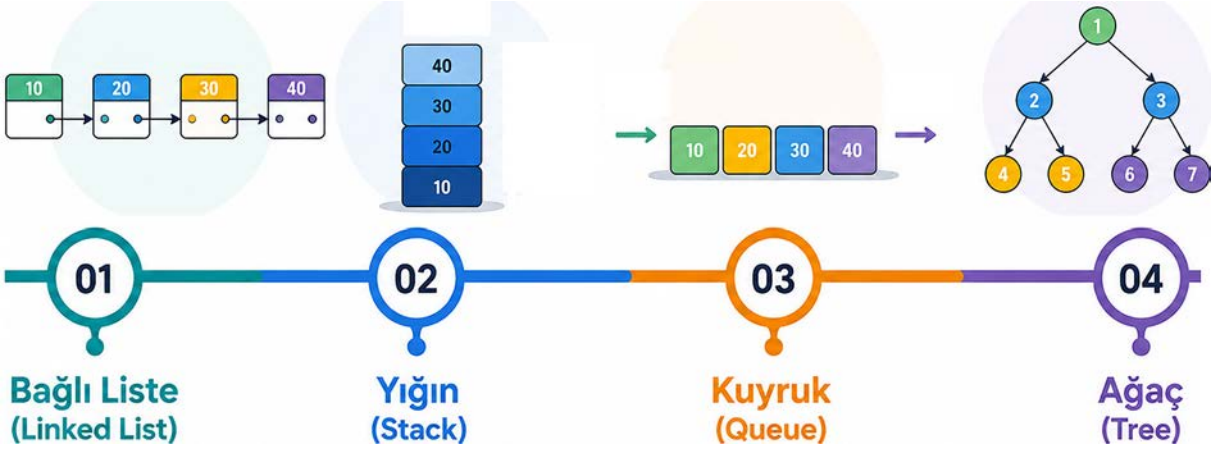
Kök (Root): Ağaç yapısındaki en üst düğümdür. Tüm diğer düğümler kökten aşağıya doğru dallanır.

LIFO: Son Giren İlk Çıkar (Last In First Out). Sisteme en son eklenen verinin, geri çağırma işleminde ilk sırada yer almasını sağlayan tersine veri akış mekanizmasıdır.

FIFO: İlk Giren İlk Çıkar (First In First Out). Verilerin belleğe geliş sırasını koruyarak ilk eklenen verinin yine ilk olarak işleme alındığı doğrusal sıralama kuralıdır.

Karmaşıklık (Complexity): Bir işlemin ne kadar süre veya bellek gerektirdiğini ifade eden ölçüttür. O(n) gösterimi ile ifade edilir.

Python'da kullanılan temel veri yapıları Görsel 9.1'de gösterilmiştir.



Görsel 9.1: Temel veri yapıları

Doğrusal Veri Yapıları

Verilerin birbirlerine **sıralı** biçimde bağlı olduğu veri yapılarına **doğrusal veri yapısı** (linear data structure) denir. Her eleman en fazla bir önceki ve bir sonraki elemana bağlanır. Veriler bellekte art arda ya da bağlantılar aracılığıyla sıralı şekilde tutulur. Tablo 9.1'de programlama dillerinde tanımlanabilen doğrusal veri yapı türleri ve günlük hayattaki karşılıkları listelenmiştir.

Tablo 9.1: Doğrusal Veri Yapı Türleri

Yapı	Açıklama	Günlük Hayat Örneği
Dizi (Array)	Aynı türdeki verilerin ardışık bellek hücrelerinde saklanması	Markette işlem sırası
Bağlı Liste (Linked List)	Her düğümün bir sonraki elemanı işaret ettiği zincir yapısı	Müzik çalardaki şarkı listesi
Yığın (Stack)	Son giren ilk çıkar (LIFO) prensibiyle çalışan yapı	Tarayıcı geri butonu
Kuyruk (Queue)	İlk giren ilk çıkar (FIFO) prensibiyle çalışan yapı	Yazıcı kuyruğu

Hiyerarşik Veri Yapıları

Verilerin üst alt ilişkisiyle düzenlendiği yapıya **hiyerarşik veri yapısı** (hierarchical data structure) denir. Bu veri yapısında bir veri birden fazla veriyle bağlantılı olabilir. Böylece ağaç veya grafik biçiminde karmaşık ilişkiler modellenebilir. Tablo 9.2'de programlama dilinde tanımlanabilen hiyerarşik veri yapı türleri ve günlük hayattaki karşılıkları verilmiştir.

Tablo 9.2: Hiyerarşik Veri Yapı Türleri

Yapı	Açıklama	Günlük Hayat Örneği
Ağaç (Tree)	Düğümlerden oluşan, bir kök düğümden dallanan hiyerarşik yapı	Şirket organizasyon şeması, dosya sistemi
İkili Arama Ağacı (BST)	Her düğümün en fazla iki alt düğümü olduğu, sıralı arama ağacı	Sözlük, telefon rehberi arama
AVL Ağacı	Yükseklik dengesi korunan, öz düzenlemeli ikili arama ağacı	Veri tabanı indeksleme

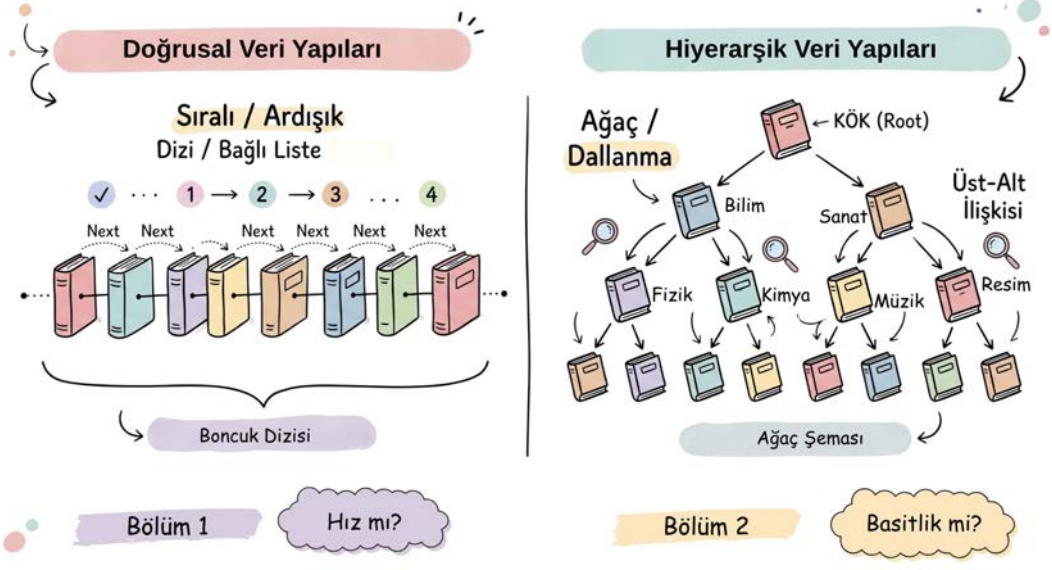
SIRA SİZDE

Aşağıda bir kütüphanedeki kitapların sınıflandırılma biçimini açıklaya iki metin ve görsel verilmiştir.

Binlerce kitabın olduğu büyük bir kütüphanede iki bölüm farklı yöntemlerle yönetilir.

1. Bölüm: Kitaplar uzun bir raf boyunca yan yana dizilir ve yeni bir kitap geldiğinde rafın sonuna eklenir. Bir kitabı bulmak için ilk kitaptan başlanarak tek tek incelenir ve rafın sonuna kadar ilerlenir ancak rafa yeni bir kitap eklemek oldukça kolaydır.

2. Bölüm: Buradaki girişte uzun bir raf yoktur sadece bir tabela bulunur. Tabelada iki ok işareti vardır ve üzerinde, "Bilim için sağdaki odaya, sanat için soldaki odaya gidin." yazısı yer alır. Her odada yeni bir ayrım yapılır ve odalardan ilerlenerek aranılan kitaba ulaşılır. Kitabı bulmak hızlıdır ama yeni bir kitap geldiğinde onu doğru dala yerleştirmek için kütüphane görevlisinin epeyce ter dökmesi gerekir.



Verilen bilgilere göre aşağıdaki soruları cevaplayınız.

1. Elinizde sadece on kitap varsa hangi yöntemle arama yaparsınız? Peki on milyon kitap olduğunda hangi yöntemi kullanırsınız?

.....

2. Koridordaki kitapları tek tek kaydırdığınızı (dizide araya eleman ekleme) düşününüz. Bu durum kütüphaneciyi neden yorar?

.....

3. Telefon rehberinde 1. bölümdeki yöntem mi yoksa 2. bölümdeki yöntem mi kullanılır? Neden?

.....

4. Siz kütüphaneci olsaydınız tüm kitapları 2. bölümdeki yapıya taşımak için harcanan zamana değeceğini düşünür müydünüz? Hangi yöntemi tercih ederdiniz?

.....

BİLGİ KUTUSU

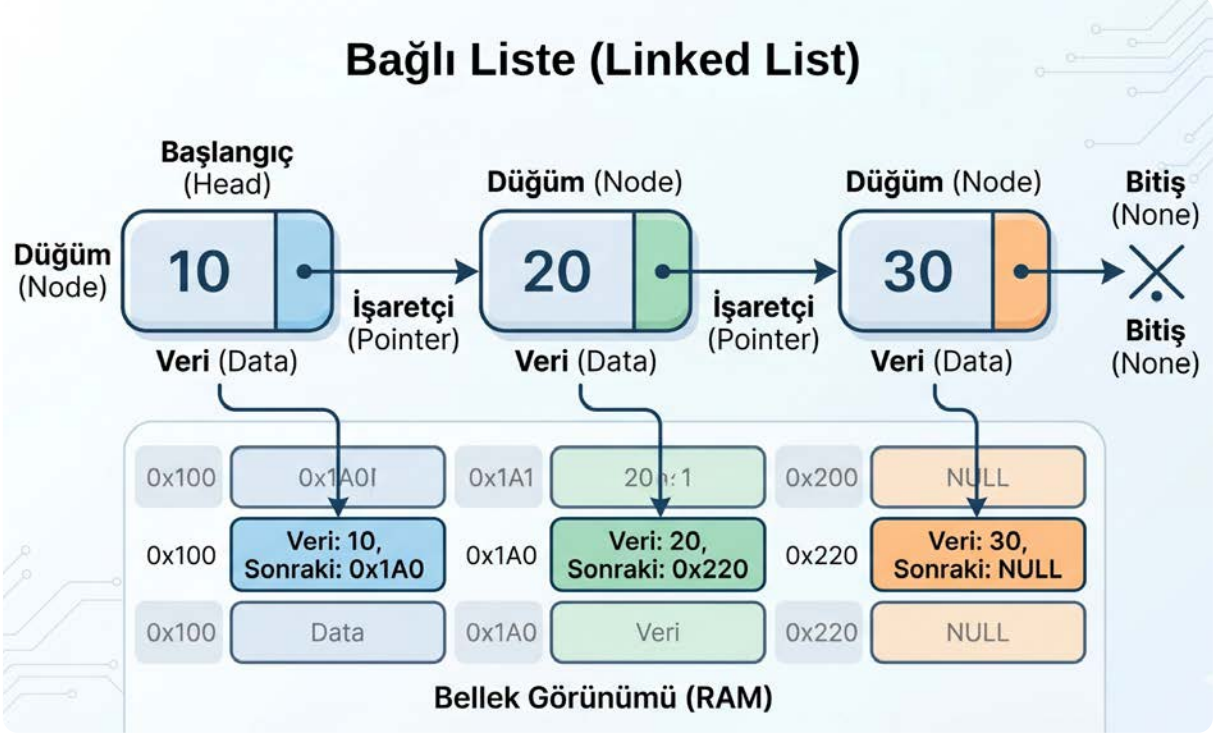
Programlama dili liste, demet, küme ve sözlük gibi yerleşik veri yapıları sunar. Bu öğrenme birimindeki bağlı liste, yığın, kuyruk ve ağaç yapıları doğrudan yerleşik olarak bulunmaz. Nesne yönelimli programlama yapısı kullanılarak sıfırdan inşa edilir. Bu yaklaşım veri yapılarının iç işleyişini kavramsal düzeyde anlamayı sağlar.

Bağlı Liste

Her verinin bir sonraki veriye işaret ettiği ve **düğümlerden** (node) oluşan **doğrusal** veri yapısına **bağlı liste** (linked list) denir. Programlama dilinin yerleşik liste yapısından farklı olarak bağlı listede veriler bellekte ardışık hücrelerde bulunmak zorunda değildir. Her düğüm veriyi ve bir sonraki düğümün adresini birlikte taşır.

Arkadaşınıza bir zarf verdiğinizi düşününüz. İçinde ilk olarak ipucu yer alır. Buna **başlangıç** (head) denir. Zarfın içinde bir görev (veri) ve bir sonraki ipucunun nerede olduğu (adres) yazılıdır. İlk zarf "Kütüphaneye git." der. Kütüphaneye gidildiğinde başka bir zarf bulunur. "Kitabı oku ve sonra laboratuvara git." yazar. Araya bir kantin durağı eklenmek istenirse binaların yerini değiştirmek gerekmez. Kütüphanedeki notun "Kantine git." olarak güncellenmesi yeterlidir.

Bağlı listenin temel yapı taşı düğümdür. Her düğüm, **veri** (data) ve bir sonraki düğümü gösteren **işaretçi** (next) bileşenlerini barındırır. Görsel 9.2'de bağlı listenin yapısı ve ana bellekte verilerin yerleşimi gösterilmektedir. Programlama dilinde düğüm **sınıf** olarak tanımlanır.



Görsel 9.2: Bağlı listenin yapısı ve ana bellekte verilerin yerleşimi

1. ÖRNEK UYGULAMA

Amaç: Dugum() sınıfı tanımlamak ve bu sınıftan düğüm oluşturmak

```
1   class Dugum:
2       def __init__(self, veri):
3           self.veri = veri      # Düğümün taşıdığı veri
4           self.sonraki = None   # Bir sonraki düğümün adresi
5
6   # Düğüm oluşturma
7   d1 = Dugum(10)
8   d2 = Dugum(20)
9   d3 = Dugum(30)
10
11  # Düğümleri birbirine bağlama
12  d1.sonraki = d2
13  d2.sonraki = d3
14
15  # Listeyi gezerek yazdırma
16  guncel = d1
17  while guncel is not None:
18      print(guncel.veri, end=' -> ')
19      guncel = guncel.sonraki
20  print('None')
```

EKRAN ÇIKTISI

```
10 -> 20 -> 30 -> None
```

BİLGİ KUTUSU

None, programlama dilinde **hiçbir şeyi** temsil eden özel bir değerdir. Bağlı listede son düğümün **sonraki** alanı **None** olarak atanır ve listenin sonuna gelindiği anlaşılır. **while guncel is not None:** döngüsü, None ile karşılaşıldığında durur.

Tek Yönlü Bağlı Liste

Her düğümün yalnızca bir sonraki düğümü gösterdiği bağlı liste türüne **tek yönlü bağlı liste** (singly linked list) denir. Önceki veriye doğru ilerlemek mümkün değildir. Listenin başını gösteren **başlık** (head) referansı tüm listeye erişim noktasıdır. Tek yönlü bağlı liste üzerinde yapılabilecek temel işlemler başa ekleme, sona ekleme, ortaya ekleme, silme ve **gezme** (traversal) işlemleridir.

2. ÖRNEK UYGULAMA

Amaç: Tek yönlü bağlı liste oluşturmak

```
1   class Dugum:
2       def __init__(self, veri):
3           self.veri = veri
4           self.sonraki = None
5   class BagliListe:
6       def __init__(self):
7           self.bas = None          # Listenin ilk düğümü
8       def basa_ekle(self, veri):
9           yeni = Dugum(veri)
10          yeni.sonraki = self.bas  # Yeni düğüm eski başı gösterir
11          self.bas = yeni          # Baş referansı güncellenir
12      def sona_ekle(self, veri):
13          yeni = Dugum(veri)
14          if self.bas is None:     # Liste boşsa baş olur
15              self.bas = yeni
16              return
17          guncel = self.bas
18          while guncel.sonraki:    # Sona kadar git
19              guncel = guncel.sonraki
20          guncel.sonraki = yeni    # Son düğüme bağla
21      def listele(self):
22          guncel = self.bas
23          while guncel:
24              print(guncel.veri, end=' -> ')
25              guncel = guncel.sonraki
26          print('None')
27  # Test
28  liste = BagliListe()
29  liste.sona_ekle(10)
30  liste.sona_ekle(20)
31  liste.sona_ekle(30)
32  liste.basa_ekle(5)
33  liste.listele()
```

EKRAN ÇIKTISI

5 -> 10 -> 20 -> 30 -> None

SIRA SİZDE

2. örnek uygulamada liste.basa_ekle() ve liste.sona_ekle() fonksiyonlarını rastgele ekleyeceğiniz on sayı ile değiştiriniz. Girdiğiniz verilere göre listede ne tür değişiklikler oldu? Bu değişikliklerin neden gerçekleştiğine dair düşüncelerinizi gerekçeleri ile yazınız.

.....

.....

3. ÖRNEK UYGULAMA

Amaç: Tek yönlü bağlı listede bir veriyi listeden silmek

1. Adım: 2. örnek uygulama kopyalanır.

2. Adım: sil() fonksiyonu eklenir.

3. Adım: Bağlı listede bulunan ve bulunmayan birer verinin silinmesi test edilir.

```
32      def sil(self, veri):
33          if self.bas is None:
34              print('Liste boş!')
35              return
36          # Silinecek düğüm baştaysa
37          if self.bas.veri == veri:
38              self.bas = self.bas.sonraki
39              return
40          # Ortada veya sonda arama
41          guncel = self.bas
42          while guncel.sonraki:
43              if guncel.sonraki.veri == veri:
44                  guncel.sonraki = guncel.sonraki.sonraki
45                  return
46              guncel = guncel.sonraki
47          print(f'{veri} listede bulunamadı!')
48  # Test
49  liste = BagliListe()
50  liste.sona_ekle(10)
51  liste.sona_ekle(20)
52  liste.sona_ekle(30)
53  liste.basa_ekle(5)
54  liste.sil(20)
55  liste.sil(15)
56  liste.listele()
```

EKRAN ÇIKTISI

```
15 listede bulunamadı!
5 -> 10 -> 30 -> None
```

Çift Yönlü Bağlı Liste

Her düğümün iki işaretçi taşıdığı bağlı liste türüne **çift yönlü bağlı liste** (doubly linked list) denir. Bir sonraki düğümü gösteren sonraki (next) ve bir önceki düğümü gösteren önceki (prev) işaretçisi bulunur. Böylece listede ileri geri hareket etmek mümkün olur. Bir müzik çalardaki **önceki şarkı** ve **sonraki şarkı** butonları çift yönlü bağlı liste mantığıyla çalışır.

4. ÖRNEK UYGULAMA

Amaç: Çift yönlü bağlı liste oluşturmak

```python
class Dugum:
    def __init__(self, veri):
        self.veri    = veri
        self.sonraki = None
        self.onceki  = None   # Çift yönlü: önceki düğüm

class CiftYonluListe:
    def __init__(self):
        self.bas = None
        self.son = None       # Kuyruk referansı

    def sona_ekle(self, veri):
        yeni = Dugum(veri)
        if self.bas is None:
            self.bas = self.son = yeni
            return
        yeni.onceki  = self.son
        self.son.sonraki = yeni
        self.son = yeni

    def ileri_listele(self):
        guncel = self.bas
        print('İleri:', end=' ')
        while guncel:
            print(guncel.veri, end=' -> ')
            guncel = guncel.sonraki
        print('None')

    def geri_listele(self):
        guncel = self.son
        print('Geri: ', end=' ')
        while guncel:
            print(guncel.veri, end=' -> ')
            guncel = guncel.onceki
        print('None')

# Test
liste = CiftYonluListe()
liste.sona_ekle(10)
liste.sona_ekle(20)
liste.sona_ekle(30)
liste.ileri_listele()
liste.geri_listele()
```

EKRAN ÇIKTISI

```
İleri: 10 -> 20 -> 30 -> None
Geri:  30 -> 20 -> 10 -> None
```

Dairesel Bağlı Liste

Son düğümün işaretçisinin None yerine listenin ilk düğümünü (head) gösteren bağlı liste türüne **dairesel bağlı liste** (circular linked list) denir. Böylece liste sonsuz bir döngü oluşturur. Müzik çalarda "Liste bitince başa al." modu bu yapıyla modellenebilir.

5. ÖRNEK UYGULAMA

Amaç: Dairesel bağlı liste oluşturmak

```
class Dugum:
    def __init__(self, veri):
        self.veri    = veri
        self.sonraki = None
class DaireselListe:
    def __init__(self):
        self.bas = None
    def ekle(self, veri):
        yeni = Dugum(veri)
        if self.bas is None:
            self.bas = yeni
            yeni.sonraki = self.bas   # Kendine bağla
            return
        guncel = self.bas
        while guncel.sonraki != self.bas:
            guncel = guncel.sonraki
        guncel.sonraki = yeni
        yeni.sonraki   = self.bas     # Son düğüm başa bağlanır
    def listele(self, tur_sayisi=1):
        if self.bas is None:
            return
        guncel = self.bas
        sayac  = 0
        toplam = tur_sayisi * self._uzunluk()
        while sayac < toplam:
            print(guncel.veri, end=' -> ')
            guncel = guncel.sonraki
            sayac += 1
        print('... (döngü)')
    def _uzunluk(self):
        if self.bas is None: return 0
        sayac, guncel = 1, self.bas.sonraki
        while guncel != self.bas:
            sayac += 1
            guncel = guncel.sonraki
        return sayac
# Test
sarki_listesi = DaireselListe()
sarki_listesi.ekle('Şarkı A')
sarki_listesi.ekle('Şarkı B')
sarki_listesi.ekle('Şarkı C')
sarki_listesi.listele(tur_sayisi=2)
```

EKRAN ÇIKTISI

Şarkı A -> Şarkı B -> Şarkı C -> Şarkı A -> Şarkı B -> Şarkı C -> ... (döngü)

Bağlı liste türlerinin özellikleri ve kullanım alanları Tablo 9.3'te listelenmiştir.

Tablo 9.3: Bağlı Liste Türlerinin Karşılaştırılması

Özellik	Tek Yönlü	Çift Yönlü	Dairesel
Yön	Yalnızca ileri	İleri ve geri	Döngüsel (sonsuz)
İşaretçi Sayısı	1 (next)	2 (next + prev)	1 (~~next~~ head)
Bellek Kullanımı	Az	Fazla	Az
Kullanım Alanı	Dinamik bellek yönetimi	Geri al / ileri git	Döngüsel liste işlemleri
Program Örneği	Dinamik bellek	Tarayıcı geçmişi	Şarkı listesinin kendini tekrarlaması

1. UYGULAMA

Bağlı Listeyle Uygulama Geliştiriyorum

Süre: 40 dakika

Amaç: Çift yönlü bağlı liste kullanarak bir CalmaListesi() tasarlamak

Yönerge: Yazılan adımları takip ederek uygulamayı tamamlayınız.

1. Adım: Çift yönlü bağlı liste kullanarak bir CalmaListesi() sınıfı tasarlayınız.

2. Adım: Sınıfa aşağıdaki metotları ekleyiniz.

- sarki_ekle(ad, sanatci): Listeye yeni şarkı ekler.
- sarki_sil(ad): Belirtilen şarkıyı listeden siler.
- sonraki_sarki(): Bir sonraki şarkıya geçer, güncel şarkıyı ekrana yazar.
- onceki_sarki(): Bir önceki şarkıya geçer, güncel şarkıyı ekrana yazar.
- listeyi_goster(): Tüm şarkıları sırayla ekrana yazar.

3. Adım: En az beş şarkı ekleyerek tüm metotları test ediniz.

4. Adım: Programınızın ekran çıktısını kaydediniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

SIRA SİZDE

Bağlı listelerin, programlama dilinin yerleşik liste (list) yapısından farklı olduğu **iki temel durumu** örneklerle açıklayınız. Hangi durumlarda bağlı liste daha avantajlı, hangi durumlarda programlama dilinin listesi daha kullanışlıdır? Gerekçeleriyle yazınız.

.....

.....

DİKKAT

Bağlı listede belirli bir indeksteki elemana erişmek için listenin başından itibaren sırayla ilerlemek gerekir. Örneğin 100. elemana ulaşmak için 99 düğümü gezmek zorunludur. Bu durum, listeye kıyasla erişimi yavaşlatır. Bağlı liste, ekleme ve silme işlemlerinde hızlı, rastgele erişimde yavaştır.

Yığın

Verilerin yalnızca bir uçtan eklenip çıkarıldığı, **son giren ilk çıkar** (last in first out LIFO) prensibine göre çalışan doğrusal veri yapısına **yığın** (stack) denir.

Mutfakta üst üste konmuş tabaklardan biri alınmak istediğinde en üstteki tabak alınır ve yeni tabaksa yine en üste konur. En alta konan tabağa en son ulaşılır. Yığın veri yapısı da bu şekilde çalışır. Tablo 9.4'te yığın veri yapısının işlemleri ve açıklamaları listelenmiştir.

Tablo 9.4: Yığın Veri Yapısının Temel İşlemleri

İşlem	Açıklama	Gerçek Hayat Karşılığı
push(eleman)	Yığının tepesine yeni eleman ekler.	Tabak yığınına yeni tabak koymak
pop()	Yığının tepesindeki elemanı çıkarır ve döndürür.	Yığından en üstteki tabağı almak
peek()	Tepedeki elemanı gösterir, çıkarmaz.	En üstteki tabağa bakıp almamak
is_empty()	Yığının boş olup olmadığını kontrol eder.	Tabak kaldı mı diye bakmak
size()	Yığındaki eleman sayısını döndürür.	Kaç tabak olduğunu saymak

6. ÖRNEK UYGULAMA

Amaç: Yigin() sınıfı oluşturarak metotlarını yazmak

```
class Dugum:
    def __init__(self, veri):
        self.veri    = veri
        self.sonraki = None
class Yigin:
    def __init__(self):
        self.tepe   = None   # Yığının tepesi
        self.boyut  = 0
    def push(self, veri):
        yeni          = Dugum(veri)
        yeni.sonraki  = self.tepe   # Yeni düğüm eski tepeyi gösterir
        self.tepe     = yeni        # Tepe güncellenir
        self.boyut   += 1
        print(f'{veri} yığına eklendi.')
    def pop(self):
        if self.is_empty():
            print('Hata: Yığın boş!')
            return None
        cikarilan    = self.tepe.veri
        self.tepe    = self.tepe.sonraki
        self.boyut  -= 1
        return cikarilan
    def peek(self):
        if self.is_empty():
            return None
        return self.tepe.veri
    def is_empty(self):
        return self.tepe is None
    def size(self):
        return self.boyut
# Test
y = Yigin()
y.push('Sayfa1')
y.push('Sayfa2')
y.push('Sayfa3')
print('Tepedeki:', y.peek())
print('Çıkarılan:', y.pop())
print('Tepedeki:', y.peek())
print('Boyut:', y.size())
print('Yığın boş mu?', y.is_empty())
```

EKRAN ÇIKTISI

```
Sayfa1 yığına eklendi.
Sayfa2 yığına eklendi.
Sayfa3 yığına eklendi.
Tepedeki: Sayfa3
Çıkarılan: Sayfa3
Tepedeki: Sayfa2
Boyut: 2
Yığın boş mu? False
```

BİLGİ KUTUSU

Yığın veri yapısı yerleşik liste kullanılarak da oluşturulabilir. **append()** metodu veriyi en sona ekler, **pop()** metodu ise veri çıkarma işlevi görür. Gerçek projelerde yerleşik **collections.deque** yapısı hem yığın hem kuyruk için önerilir.

SIRA SİZDE

Web tarayıcının adres çubuğunda geri ve ileri butonları bulunur. Bu butonlar yığın yapısını kullanır. **Yalnızca bir yığın kullanılarak** geri ileri butonunun çalışması mümkün müdür? Cevabınızı gerekçesiyle yazınız.

.....

.....

.....

Kuyruk

Verilerin bir uçtan eklenip diğer uçtan çıkarıldığı **ilk giren ilk çıkar** (first in first out FIFO) prensibine göre çalışan doğrusal veri yapısına **kuyruk** (queue) denir.

Markette kasa sırasında ilk gelen müşteri önce hizmet alır. Yeni gelen müşteri sıranın en arkasına geçer ve müşteriler sırayı atlayamaz. Kuyruk veri yapısı bu mantıkla çalışır. Tablo 9.5'te kuyruk veri yapısının işlemleri ve açıklamaları listelenmiştir.

Tablo 9.5: Kuyruk Veri Yapısının Temel İşlemleri

İşlem	Açıklama	Gerçek Hayat Karşılığı
enqueue(eleman)	Kuyruğun sonuna yeni eleman ekler.	Sıranın arkasına geçmek
dequeue()	Kuyruğun önündeki elemanı çıkarır ve döndürür.	Sıradaki ilk müşteriye hizmet vermek
front()	Kuyruğun önündeki elemanı gösterir, çıkarmaz.	Sıranın başındaki kişiye bakmak
is_empty()	Kuyruğun boş olup olmadığını kontrol eder.	Sırada kimse var mı diye bakmak
size()	Kuyruktaki eleman sayısını döndürür.	Sırada kaç kişi var diye saymak

7. ÖRNEK UYGULAMA

Amaç: Bağlı liste kullanan bir Kuyruk() sınıfı oluşturmak

```
class Dugum:
    def __init__(self, veri):
        self.veri    = veri
        self.sonraki = None
class Kuyruk:
    def __init__(self):
        self.on    = None   # Kuyruğun önü (çıkış)
        self.arka  = None   # Kuyruğun arkası (giriş)
        self.boyut = 0
    def enqueue(self, veri):
        yeni = Dugum(veri)
        if self.arka is None:
            self.on = self.arka = yeni
        else:
            self.arka.sonraki = yeni
            self.arka         = yeni
        self.boyut += 1
        print(f'{veri} kuyruğa eklendi.')
    def dequeue(self):
        if self.is_empty():
            print('Hata: Kuyruk boş!')
            return None
        cikarilan = self.on.veri
        self.on   = self.on.sonraki
        if self.on is None:
            self.arka = None   # Kuyruk tamamen boşaldı
        self.boyut -= 1
        return cikarilan
    def front(self):
        if self.is_empty():
            return None
        return self.on.veri
    def is_empty(self):
        return self.on is None
    def size(self):
        return self.boyut
k = Kuyruk()
k.enqueue('Belge_1.pdf')
k.enqueue('Rapor.docx')
k.enqueue('Fatura.xlsx')
print('Sıradaki:', k.front())
print('İşlenen:', k.dequeue())
print('İşlenen:', k.dequeue())
print('Kalan belge sayısı:', k.size())
```

EKRAN ÇIKTISI

```
Belge_1.pdf kuyruğa eklendi.
Rapor.docx kuyruğa eklendi.
Fatura.xlsx kuyruğa eklendi.
Sıradaki: Belge_1.pdf
İşlenen: Belge_1.pdf
İşlenen: Rapor.docx
Kalan belge sayısı: 1
```

Yığın ve kuyruk yapılarının karşılaştırması Tablo 9.6'da listelenmiştir.

Tablo 9.6: Yığın ve Kuyruk Karşılaştırması

Özellik	Yığın (Stack)	Kuyruk (Queue)
Prensip	LIFO (Son giren ilk çıkar.)	FIFO (İlk giren ilk çıkar.)
Ekleme Noktası	Yapının başına	Yapının sonuna
Çıkarma Noktası	Yapının başından	Yapının başından
Temel Metotlar	push(), pop(), peek()	enqueue(), dequeue(), front()
Kullanım Alanları	Geri al, parantez denetimi, çağrı yığını	Yazıcı kuyruğu, banka sırası, ağ paketleri
Benzetme	Tabak yığını	Kasa sırası

2. UYGULAMA

Yığın ve Kuyruk Veri Yapılarıyla Program Geliştiriyorum

Süre: 40 + 40 dakika

Amaç: Bir hesap makinesi uygulamasında yığın ve kuyruk veri yapılarını kullanmak

Yönerge: Yazılan adımları takip ederek uygulamayı tamamlayınız.

1. Adım: Bir hesap makinesi uygulaması için basit bir hesaplama geçmişi yöneticisi geliştirilir. Uygulama; yapılan hesaplamaları yığın ve kuyruk yapılarında saklamalı, son hesaplamayı geri alabilmeli ve tüm geçmişi ekrana yazdırabilmelidir.

Yigin() sınıfını (push, pop, peek, is_empty, size) bağlı liste ile yazınız.

2. Adım: HesapGecmisi() sınıfını oluşturarak şu metotları ekleyiniz:

- hesap_ekle(ifade, sonuc): İfadeyi ve sonucu yığına ekler.
- son_hesabi_geri_al(): En son eklenen hesabı yığından çıkarır.
- gecmisi_goster(): Tüm hesapları son yapılandan ilk yapılana doğru listeler.

3. Adım: Kuyruk() sınıfını (enqueue, dequeue, front, is_empty, size) bağlı liste ile yazınız.

4. Adım: En az dört farklı hesaplama ekleyiniz (toplama, çıkarma, çarpma, bölme).

5. Adım: İki kez geri al işlemi yapınız ve geçmişin nasıl değiştiğini gözlemleyiniz.

6. Adım: Kuyruk yapısı ile hesaplama işlemlerini sırasıyla görüntüleyiniz.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.



Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



Ağaç Yapıları

Bağlı liste, yığın ve kuyruk yapıları doğrusal yapılardır. Her eleman bir önceki ve bir sonrakiyle bağlıdır. Günlük hayatta bazı ilişkiler doğrusal değil hiyerarşiktir. Şirketin organizasyon şeması, dosya sistemi, aile soy ağacı ya da web sitesinin sayfa yapısı hiyerarşik ilişkilere örnektir. Bu tür ilişkileri modellemek için **ağaç** (tree) veri yapısı kullanılır.

Bir şirketin organizasyon şemasında en tepede genel müdür bulunur. Genel müdüre bağlı birkaç bölüm müdürü vardır ve her müdürün altında da kendi ekipleri yer alır (Görsel 9.3). Tek bir kök (genel müdür) ve ondan dallanan alt düğümlerden oluşan hiyerarşik diyagram **ağaç yapısını** tanımlar. Tablo 9.7'de ağaç yapılarında kullanılan kavramlar ve açıklamaları listelenmiş,



Görsel 9.3: Ağaç yapısı ve bileşenleri

Tablo 9.7: Ağaç Yapıları Kavramları

Kavram	Açıklama	Örnek
Kök (Root)	Ağacın en tepesindeki tek düğüm Üst düğümü yoktur.	Genel Müdür
Düğüm (Node)	Veri tutan her bir eleman	Her çalışan
Kenar (Edge)	İki düğümü birbirine bağlayan bağlantı	Yönetici–ast ilişkisi
Yaprak (Leaf)	Alt düğümü olmayan uç düğümler	Astı olmayan çalışan
Ebeveyn (Parent)	Kendisine bağlı alt düğümleri olan düğüm	Bölüm müdürü
Çocuk (Child)	Bir ebeveyn düğüme bağlı alt düğüm	Bölüm müdürünün ekip üyeleri
Yükseklik (Height)	Kökten en uzak yaprağa kadar olan kenar sayısı	Şirketteki hiyerarşi derinliği
Seviye (Level)	Kökten itibaren bir düğümün derinlik katmanı Kök seviye 0'dır.	Yönetim kademesi
Alt Ağaç (Subtree)	Herhangi bir düğüm ve ona bağlı tüm torunlarından oluşan yapı	Bir bölüm ve tüm ekibi

İkili Ağaç

Düğümlerden her biri için en fazla **iki alt düğüm barındırabilen** özel ağaç türüne **ikili ağaç** (binary tree) denir. Bu iki alt düğüm **sol çocuk** (left child) ve **sağ çocuk** (right child) olarak adlandırılır.

8. ÖRNEK UYGULAMA

Amaç: IkiliDugum() sınıfı ile ikili ağaç oluşturmak

```
1   class IkiliDugum:
2       def __init__(self, veri):
3           self.veri  = veri
4           self.sol   = None    # Sol çocuk
5           self.sag   = None    # Sağ çocuk
6   # Ağaç kurulumu
7   #         10
8   #        /  \
9   #       5    15
10  #      / \     \
11  #     3   7     20
12  kok = IkiliDugum(10)
13  kok.sol         = IkiliDugum(5)
14  kok.sag         = IkiliDugum(15)
15  kok.sol.sol     = IkiliDugum(3)
16  kok.sol.sag     = IkiliDugum(7)
17  kok.sag.sag     = IkiliDugum(20)
18
19  # Kök ve çocuklarını yazdırma
20  print('Kök           :', kok.veri)
21  print('Sol çocuk     :', kok.sol.veri)
22  print('Sağ çocuk     :', kok.sag.veri)
23  print('Sol→Sol       :', kok.sol.sol.veri)
24  print('Sol→Sağ       :', kok.sol.sag.veri)
25  print('Sağ→Sağ       :', kok.sag.sag.veri)
```

EKRAN ÇIKTISI

```
Kök           : 10
Sol çocuk     : 5
Sağ çocuk     : 15
Sol → Sol     : 3
Sol → Sağ     : 7
Sağ → Sağ     : 20
```

İkili Arama Ağacı

Her düğümün değeri, sol alt ağaçtaki tüm değerlerden büyük, sağ alt ağaçtaki tüm değerlerden küçük olan özel ikili ağaç türüne **ikili arama ağacı** (binary search tree BST) adı verilir. Bu kural; ağaçta çok hızlı arama, ekleme ve silme işlemlerinin yapılmasını sağlar.

İkili arama ağacı kuralı; Sol < Kök < Sağ. Sözlük veya telefon rehberinde bir isim ararken A harfinden başlanarak ortaya gidilir oradan da hedefe ulaşılır.

9. ÖRNEK UYGULAMA

Amaç: BST() sınıfı ile eleman **ekleme** ve **arama** işlemlerini gerçekleştirmek

```
class DugumBST:
    def __init__(self, veri):
        self.veri = veri
        self.sol  = None
        self.sag  = None
class BST:
    def __init__(self):
        self.kok = None
    def _ekle(self, dugum, veri):
        #Özyinelemeli ekleme
        if dugum is None:
            return DugumBST(veri)       # Yeni düğüm oluştur
        if veri < dugum.veri:
            dugum.sol = self._ekle(dugum.sol, veri)  # Sola git
        elif veri > dugum.veri:
            dugum.sag = self._ekle(dugum.sag, veri)  # Sağa git
        return dugum
    def ekle(self, veri):
        self.kok = self._ekle(self.kok, veri)
    def _ara(self, dugum, veri):
        #Özyinelemeli arama
        if dugum is None:
            return False                # Bulunamadı
        if veri == dugum.veri:
            return True                 # Bulundu
        if veri < dugum.veri:
            return self._ara(dugum.sol, veri)  # Sola git
        return self._ara(dugum.sag, veri)      # Sağa git
    def ara(self, veri):
        return self._ara(self.kok, veri)
    def ic_sira(self, dugum):
        if dugum:
            self.ic_sira(dugum.sol)
            print(dugum.veri, end=' ')
            self.ic_sira(dugum.sag)
agac = BST()
for sayi in [50, 30, 70, 20, 40, 60, 80]:
    agac.ekle(sayi)
print('İç sıra (sıralı):', end=' ')
agac.ic_sira(agac.kok)
print()
print('60 var mı?', agac.ara(60))
print('45 var mı?', agac.ara(45))
```

EKRAN ÇIKTISI

```
İç sıra (sıralı): 20 30 40 50 60 70 80
60 var mı? True
45 var mı? False
```

AVL Ağacı

İkili arama ağacında elemanlar sıralı şekilde eklendiğinden ağaç, tek tarafa doğru uzayabilir ve bağlı bir listeye dönüşebilir. Bu durumda arama, ekleme ve silme işlemleri yavaşlar. Bu sorunu çözmek için **AVL ağacı** geliştirilmiştir. Her ekleme veya silme işleminden sonra dalların boyu kontrol edilir. Bir taraf çok uzarsa **döndürme** adı verilen bir yöntemle ağaç kendini yeniden dengeler. Böylece ağaç her zaman toplu ve kısa kalır. İşlemler her zaman en yüksek hızda çalışır. AVL ağacının özellikleri ve açıklamaları Tablo 9.8'de listelenmiştir.

Tablo 9.8: AVL Ağacının Özellikleri

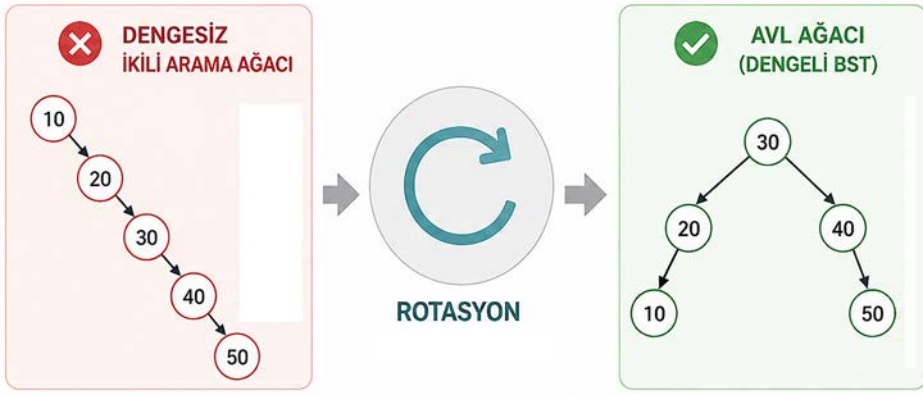
Özellik	Açıklama
Denge Faktörü	Her düğüm için: sol alt ağaç yüksekliği - sağ alt ağaç yüksekliği. Bu değer –1, 0 veya +1 olmalıdır.
Otomatik Dengeleme	Ekleme veya silme sonrasında denge bozulursa ağaç otomatik rotasyon yaparak kendini dengeler.
Rotasyon Türleri	Sola rotasyon (left rotation), sağa rotasyon (right rotation), sol-sağ (left-right), sağ-sol (right-left) olmak üzere dört rotasyon türü vardır.
Zaman Karmaşıklığı	Arama, ekleme ve silme işlemleri her zaman O(log n) sürede tamamlanır, bu BST'nin en kötü durumu olan O(n) ile karşılaştırıldığında çok daha hızlıdır.
Kullanım Alanları	Veri tabanı indeksleme, bellek yönetimi, işletim sistemi çizelgeleme

BİLGİ KUTUSU

AVL ağacı, adını 1962 yılında bu yöntemi geliştiren iki Sovyet bilim insanı Georgy **A**delson **V**elsky (Georgiy Adelson Velskiy) ve Evgenii **L**andis'ten (Yevgeniy Landis) alır.

AVL, bilgisayar bilimleri tarihinde icat edilen ilk kendi kendini dengeleyen veri yapısıdır. O güne kadar veri arttıkça bilgisayarların yavaşlaması büyük bir sorundu. Bu iki mucit, ağacın boyunu ölçüp kendini düzeltmesi fikrini matematiksel olarak kanıtladılar. Onların bu buluşuyla bugün kullanılan büyük veri tabanlarının ve hızlı arama motorlarının temeli atıldı.

AVL ağacı, dengesiz büyüyen ikili arama ağacını **rotasyon işlemleriyle** yeniden düzenleyerek arama, ekleme ve silme işlemlerinin **verimli** kalmasını sağlar (Görsel 9.4).



Görsel 9.4: Dengesiz büyüyen ikili arama ağacı ve rotasyon ile dengelenmesi

3. UYGULAMA

Ağaç Yapılarıyla Program Geliştiriyorum

Süre: 40 + 40 dakika

Amaç: Ağaç yapılarını kullanarak sözlük uygulaması geliştirmek

Yönerge: Yazılan adımları takip ederek uygulamayı tamamlayınız.

1. Adım: Ağacın kökünü belirleyiniz. Sözlüğe eklenecek ilk kelimeyi **kök düğüm** olarak seçiniz.

2. Adım: Kelime ekleme mantığı kurunuz. Yeni bir kelime eklendiğinde alfabetik sıraya bakınız. Yeni kelime mevcut kelimeden **önce** geliyorsa **sola** gidiniz. Boş bir yer bulana kadar bu işleme devam ederek kelimeyi uygun yere yerleştiriniz.

3. Adım: Arama algoritmasını çalıştırınız. Kullanıcı bir kelime sorguladığında kökten başlayınız. Aranan kelimenin kökten büyük ya da küçük olduğunu kontrol ederek her adımda ağacın bir dalına bakınız. Kelime bulunduğunda "Bulundu." mesajı veriniz. Boş bir dala ulaşılmışsa "Kelime bulunamadı." mesajı veriniz.

4. Adım: Ağacı soldan-kökten-sağa yöntemiyle dolaşarak sözlükteki tüm kelimeleri alfabetik sırada ekrana yazdırınız.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

9.2. DOSYA İŞLEMLERİ

Bir sözlük uygulaması yazarak programı çalıştırdınız ve programa yüzlerce kelime girdiniz ancak programı ya da bilgisayarı kapattığınızda girdiğiniz tüm kelimeler kaybolur. Bunun nedeni, programa girilen verilerin bilgisayarın çok hızlı çalışan ancak verileri geçici olarak saklayan **rasgele erişimli belleğe** (Random access memory RAM) kaydedilmesidir. Tekrar erişilmek istenilen verileri depolamak için bilgisayarda, **sabit disk** (harddisk) veya **SSD** (solid state drive) olarak adlandırılan depolama birimi kullanılır.

Programlama dilinde **dosya** (file) gerçek hayattaki deftere benzer. Bir deftere yazı yazılabilir sonra açıp okunabilir ve gerekirse üzerine eklemeler yapılabilir.Dosya; bilgisayarın sabit diskinde veya başka bir depolama biriminde saklanan, isimlendirilmiş veri bloğudur. Programlama dilinde genellikle **metin dosyaları** (text files) ile çalışılır. Bir dosyayla çalışmak için önce dosya **açılır** (open), çalışma tamamlanınca **kapatılır** (close).

Dosya Açma Modları

Programlama dilinde open() fonksiyonu kullanılarak dosyalar farklı modlarla açılabilir. Tablo 9.9'da dosya açma modları listelenmiştir.

Tablo 9.9: Dosya Açma Modları

Mod	Açıklama
"r" (read)	Okuma modu, dosya yoksa hata verir (Varsayılan mod).
"w" (write)	Yazma modu, dosya varsa içeriğini siler yoksa oluşturur.
"a" (append)	Ekleme modu, dosyanın sonuna veri ekler yoksa oluşturur.
"r+"	Hem okuma hem yazma işlemi yapar. Dosya mevcut olmalıdır.

BİLGİ KUTUSU

Türkçeye özgü karakterlerin (ç,ğ,ı,ş,ö,ü) dosyaya doğru kaydedilmesi ve doğru okunabilmesi için dosya açılırken **encoding="utf-8"** parametresi kullanılmalıdır.

Dosya Oluşturma ve Bilgi Yazma

Yeni bir dosya oluşturmak veya mevcut bir dosyaya yazmak için "**w**" modu kullanılır. Dosyayı **with bloğu** içinde açmak en güvenli yöntemdir. Dosyaya tek satır bilgi kaydetmek için **write()** komutu, bir liste içeriğindeki tüm bilgileri kaydetmek için **writelines()** komutu kullanılır. With bloğu bitince dosya otomatik kapanır.

10. ÖRNEK UYGULAMA

Amaç: Dosya oluşturmak ve içine veri yazmak

```
1  with open("ogrenciler.txt", "w", encoding="utf-8") as dosya:
2      dosya.write("Ali\n")
3      dosya.write("Ayşe\n")
4  print("Dosya başarıyla oluşturuldu!")
5
6  ogrenciler = ["Zeynep\n", "Can\n"]
7  with open("ogrenciler.txt", "a", encoding="utf-8") as dosya:
8      dosya.writelines(ogrenciler)
9  print("Sınıf listesi kaydedildi.")
```

EKRAN ÇIKTISI

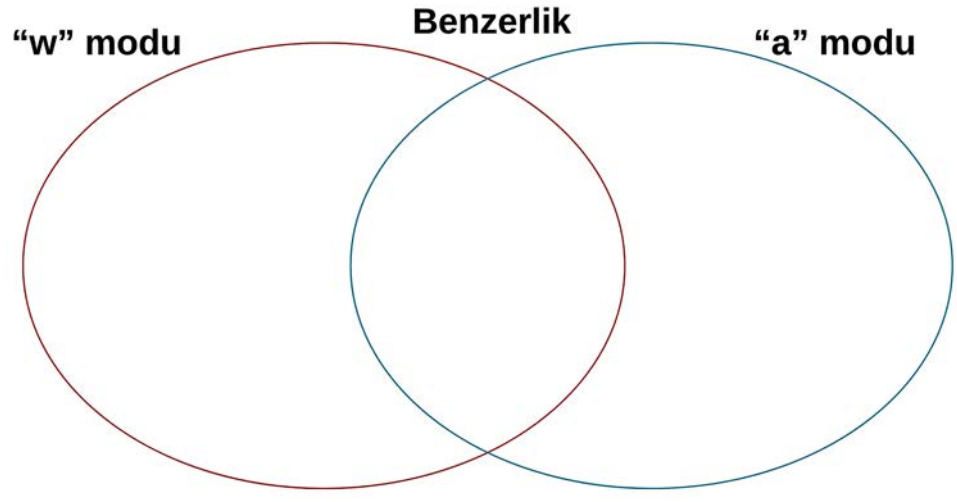
```
Dosya başarıyla oluşturuldu!
Sınıf listesi kaydedildi.
```

BİLGİ KUTUSU

\n ifadesi yeni satır (newline) karakteridir. **write()** fonksiyonu otomatik olarak satır atlamaz, her satırın sonuna **\n** eklenmezse tüm veriler tek satırda birleşir.

SIRA SİZDE

"w" ve **"a"** dosya açma modlarının benzerlik ve farklılıklarını aşağıdaki görselde uygun yerlere yazınız.



Dosya Okuma

Bir dosyanın içeriğini okumak için **"r"** modu kullanılır. Üç farklı okuma yöntemi ve açıklamaları Tablo 9.10'da listelenmiştir.

Tablo 9.10: Dosya Okuma Modları

Yöntem	Açıklama
read()	Dosyanın tüm içeriğini tek bir string olarak okur.
readline()	Dosyadan tek bir satır okur.
readlines()	Tüm satırları bir liste olarak döndürür.
for döngüsü	Dosyayı satır satır dolaşmanın en verimli yoludur.

11. ÖRNEK UYGULAMA

Amaç: Mevcut dosyadan bilgi okumak

```
1   print("1. Aşama:")
2   with open("ogrenciler.txt", "r", encoding="utf-8") as dosya:
3       icerik = dosya.read()
4       print(icerik)
5
6   print("2. Aşama:")
7   with open("ogrenciler.txt", "r", encoding="utf-8") as dosya:
8       for satir in dosya:
9           print(satir.strip())   # strip() baştaki/sondaki boşlukları temizler
10
11  print("3. Aşama:")
12  with open("ogrenciler.txt", "r", encoding="utf-8") as dosya:
13      satirlar = dosya.readlines()
14  print(f"Toplam öğrenci sayısı: {len(satirlar)}")
15  print("3. öğrenci:", satirlar[2].strip())
```

EKRAN ÇIKTISI

```
1. Aşama:
Ali
Ayşe
Zeynep
Can

2. Aşama:
Ali
Ayşe
Zeynep
Can

3. Aşama:
Toplam öğrenci sayısı: 4
3. öğrenci: Zeynep
```

SIRA SİZDE

Her çalıştırıldığında kullanıcıdan bir cümle alan ve bunu “gunluk.txt” dosyasına tarih/saatle bilgisiyle birlikte ekleyen bir program yazınız. Programı en az üç kez çalıştırarak bilgileri her defasında doğru şekilde ve tarih-saat bilgisiyle kaydedildiğini kontrol ediniz.

Dosyada Arama Yapma

Daha önce oluşturulup içine bilgi kaydedilen bir dosyada aranan veriyi bulmak için dosya satır satır okunur ve **in** operatörü kullanılır.

12. ÖRNEK UYGULAMA

Amaç: Mevcut dosyadan bilgi aramak

```
aranan = input("Aradığınız isim: ")
bulundu = False
with open("ogrenciler.txt", "r", encoding="utf-8") as dosya:
    for satir_no, satir in enumerate(dosya, start=1):
        if aranan.lower() in satir.lower():  # Büyük/küçük harf duyarsız
            print(f"Bulundu! Satır {satir_no}: {satir.strip()}")
            bulundu = True
if not bulundu:
    print(f"{aranan} dosyada bulunamadı.")
```

EKRAN ÇIKTISI

```
Aradığınız isim: ayşe
Bulundu! Satır 2: Ayşe
```

Dosyada Güncelleme veya Silme İşlemi Yapma

Metin dosyalarında doğrudan satır silme veya değiştirme mümkün değildir. Güncelleme yapmak için standart işlem adımları şu şekildedir:

1. Dosyanın tüm içeriğini oku readlines() ile okunur.
2. Okunan liste üzerinde değişiklik yapılır.
3. Dosya **"w"** modunda açılır ve güncellenmiş içerik yeniden yazılır.

13. ÖRNEK UYGULAMA

Amaç: Mevcut dosyadan bir satırı silmek

```
silinecek = input("Dosyadan silinecek öğrencinin adı: ")
# Adım 1: Oku
with open("ogrenciler.txt", "r", encoding="utf-8") as dosya:
    satirlar = dosya.readlines()
# Adım 2: Filtrele
yeni_satirlar = [s for s in satirlar if silinecek.lower() not in s.lower()]
# Adım 3: Yeniden yaz
with open("ogrenciler.txt", "w", encoding="utf-8") as dosya:
    dosya.writelines(yeni_satirlar)
fark = len(satirlar) - len(yeni_satirlar)
print(f"{fark} kayıt silindi.")
```

EKRAN ÇIKTISI

```
Dosyadan silinecek öğrencinin adı: zeynep
1 kayıt silindi.
```

Hata Yönetimi

Gerçek uygulamalarda dosya işlemleri başarısız olabilir. Dosya bulunamayabilir, erişim izni olmayabilir ya da disk dolu olabilir. Bu durumları **try-except** bloğuyla yönetmek programın çökmesini engeller.

14. ÖRNEK UYGULAMA

Amaç: Dosya işleminde hata kontrolü yapmak

```
dosya_adi = input("Okunacak dosyanın adı: ")
try:
    with open(dosya_adi, "r", encoding="utf-8") as dosya:
        icerik = dosya.read()
        print(icerik)
except FileNotFoundError:
    print(f"HATA: {dosya_adi} bulunamadı!")
except PermissionError:
    print("HATA: Bu dosyaya erişim izniniz yok.")
except Exception as e:
    print(f"Beklenmeyen hata: {e}")
else:
    print("Dosya başarıyla okundu.")
finally:
    print("İşlem tamamlandı.")
```

EKRAN ÇIKTISI

Okunacak dosyanın adı: test.txt

HATA: test.txt bulunamadı!

İşlem tamamlandı.

15. ÖRNEK UYGULAMA

Amaç: Okuma (r) ve yazma (w) modlarını birlikte kullanarak hata kontrollü bir dosya yedekleme (kopyalama) programı yazmak

```
def yedek_al(kaynak_adi, hedef_adi):
    try:
        with open(kaynak_adi, "r") as kaynak, open(hedef_adi, "w") as hedef:
            veri = kaynak.read()
            hedef.write(veri)
        print(f"[BAŞARILI] {kaynak_adi} içeriği {hedef_adi} dosyasına kopyalandı.")
    except FileNotFoundError:
        print(f"[HATA] '{kaynak_adi}' bulunamadı. Kopyalama iptal edildi.")
yedek_al("veritabani_ayarlar.txt", "yedek_ayarlar.txt")
```

EKRAN ÇIKTISI

[HATA] 'veritabani_ayarlar.txt' bulunamadı. Kopyalama iptal edildi.

4. UYGULAMA

Dosya İşlemleri Yapan Program Geliştiriyorum

Süre: 40 + 40 dakika

Amaç: Adres defteri oluşturmak

Yönerge: Yazılan adımları takip ederek uygulamayı tamamlayınız.

1. Adım: Arkadaşlarınızın ad, soyad ve telefon numaralarını tek satıra kaydedecek bir telefon rehberi uygulaması geliştiriniz. kayitlari_listele(), kayit_ekle(), kayit_ara(), kayit_sil() metodlarını yazınız.

2. Adım: İçerisinde listele, ekle, ara, sil ve çıkış seçenekleri bulunan bir menü yapısı oluşturunuz.

3. Adım: Tüm metodları test ediniz. Hatalı dosya erişiminde try-except yöntemiyle hata kontrolü yapınız.

Değerlendirme

Çalışmanız öğretmeniniz tarafından **Analitik Dereceli Puanlama Anahtarı** ile değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak analitik dereceli puanlama anahtarına ulaşabilirsiniz.

Analitik dereceli puanlama anahtarına ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

SIRA SİZDE

Aşağıdaki soruları cevaplayınız.

1. Bir hastane, hastaların tıbbi kayıtlarını .txt dosyalarında saklıyor olsaydı ne gibi sorunlar ortaya çıkabilir? Kullanıcıların yetkilendirilmesi ve güvenliği konusunda ne tür sorunlar oluşabilir?

.....

.....

.....

2. Büyük dosyalarda (milyonlarca satır) arama yaparken for döngüsü ile satır satır okumanın faydası veya zararı ne olabilir? Tüm dosyayı read() ile tek seferde okumakla farkı nedir?

.....

.....

.....

3. Güncelleme işleminde 'Oku ⟶ Değiştir ⟶ Tekrar Yaz' yönteminin riskleri neler olabilir? Program, güncelleme sırasında kapanırsa ne tür sorunlar oluşabilir?

.....

.....

.....

9.1. PROJE	
Konu	Akıllı Kütüphane Sistemini Tamamlıyorum
Kapsanan Konular	Bağlı Liste, Yığın, Kuyruk, Ağaç Yapıları, Dosya İşlemleri
Süre	40+40+40+40 dakika
Amaç	Temel veri yapılarını kapsayıcı bir projede kullanmak
Çalışma Biçimi	Bireysel

Proje Senaryosu

Bir okul kütüphanesinin kitap ödünç verme, envanter takibi ve üye kayıt işlemlerini dijitalleştirmek amacıyla **Akıllı Kütüphane Yönetim Sistemi** adında bir masaüstü yazılım geliştirildi. İlk testlerde başarıyla çalışan sistem, kütüphane ortamında aktif olarak kullanılmaya başlanmıştır.

Yazılım sahada test edilirken kullanıcılar tarafından iki kritik teknik sorun rapor edilmiştir.

1. Geliştirilen yazılımın kapatıldığında verileri kaybetmesi
2. Kitap sayısı arttıkça yaşanan yavaşlama ve donma problemi

Bu projede aşağıdaki görevleri yerine getirerek program yeni hâline güncellenecektir. Projenin sonunda hatalar giderilerek verilerin metin dosyasında güvenle saklandığı, binlerce kitap arasında bile anlık işlem yapabilen, hızlı, kararlı ve ölçeklenebilir bir kütüphane otomasyonu elde edilecektir.

Sistemin Genel Mimarisi

Projeye başlamadan önce aşağıdaki sistem analiz edilir. Her modülün hangi veri yapısını kullandığı ve modüller arasındaki ilişkiyi kavranır.

Modül	Temel İşlevler	Dosya Bağlantısı
Kitap Kataloğu	Ekle, sil, listele	katalog.txt
İşlem Geçmişi	Geri al, geçmişi göster	gecmis.txt
Ödünç Kuyruğu	Sıraya ekle, işle	odunc.txt
Arama Motoru	Ada göre hızlı ara	(bellek içi)

Her aşamayı sırayla tamamlayınız. Bir sonraki aşama önceki aşamanın çıktısını kullanacaktır. Programınızı her aşama sonunda test ederek çıktılarını kaydediniz.

1. Aşama: Kitap kataloğunu oluşturunuz.

Kapsanan İlke: Bağlı listeler

1. Görev

a) Önceki projenizde olduğu gibi bir KitapDugum() sınıfı oluşturarak bağlı liste yapısına dayalı KitapKatalogu() sınıfını tanımlayınız.
b) Kitap ekleme, silme, listeleme ve silme metotlarını tanımlayınız.
c) En az beş kitap ekleyerek listeleme metodunun çıktısını yazınız.

2. Aşama: İşlem geçmişi ve ödünç kuyruğunu oluşturunuz.

Kapsanan İlke: Yığın, kuyruk

2. Görev

a) Kütüphane sisteminde yapılan her işlem (kitap ekleme, silme, ödünç verme, iade) bir yığında saklanır. Yönetici son işlemi geri alabilir ya da tüm geçmişi görebilir. IslemGecmisi() sınıfını bağlı liste tabanlı Yigin() sınıfını kullanarak geliştiriniz.
b) Bir kitap müsait değilse talep eden öğrenciler ödünç bekleme kuyruğuna alınacaktır. Kitap iade edildiğinde kuyrukta bekleyen ilk öğrenciye otomatik bildirim gönderilecektir. OduncKuyrugu() sınıfını bağlı liste tabanlı Kuyruk() sınıfını kullanarak geliştiriniz.
c) İşlem geçmişi için neden kuyruk değil yığın kullandınız? "Geri Al" işlemi için yığının LIFO prensibi neden daha uygundur?
ç) Ödünç bekleme sistemi için neden yığın değil kuyruk kullandınız? FIFO prensibinin bu senaryodaki önemi nedir?

3. Aşama: Arama motoru oluşturunuz.

Kapsanan İlke: İkili arama ağacı

3. Görev

a) Katalogda yüzlerce kitap bulunduğunda bağlı listede arama **O(n)** zamanda gerçekleşir, tüm liste taranmak zorundadır. İkili Arama Ağacı kullanılarak kitap adına göre arama **O(log n)**'e indirilir. KitapAgaci() sınıfını yazınız. İkili arama ağacına ekleme, arama, sıralı gezme, listeleme metodlarını yazınız.
b) Katalogdaki kitapları ikili arama ağacına ekleyiniz. Ekleme sırasıyla ağacın nasıl şekillendiğini şema olarak çiziniz.

4. Aşama: Verileri .txt dosyasına kaydediniz.

Kapsanan İlke: Dosya işlemleri

4. Görev

a) Projenizin bu aşamasına kadar oluşturulan tüm veriler (katalog, işlem geçmişi, ödünç kuyruğu) yalnızca ana bellekte tutulmaktadır, program kapandığında kaybolur. Bu aşamada tüm veriler .txt dosyalarına kalıcı olarak kaydedilecek ve program her açıldığında dosyadan okunacaktır. DosyaYoneticisi() sınıfını yazınız. Bu sınıf tüm okuma ve yazma işlemlerini yönetecektir. katalog_kaydet(), katalog_yukle(), gecmis_kaydet(), gecmis_oku(), odunc_durumu_guncelle() metodlarını yazınız.
b) Program ilk kez çalıştırıldığında katalog.txt henüz oluşturulmamış olabilir. katalog_yukle() metodunda bu durumu nasıl kontrol ettiniz?

5. Aşama: Projenizi test ediniz.

Kapsanan İlke: Tüm veri yapıları entegre edilmiş şekilde

Verilen örnek test senaryosunu **kendi programınıza** göre adım adım kodlayınız.

Test Senaryosu

```
class KutuphaneSistemi:
    def __init__(self):
        self.katalog  = KitapKatalogu()
        self.gecmis   = IslemGecmisi()
        self.kuyruk   = OduncKuyrugu()
        self.agac     = KitapAgaci()
        self.dosya    = DosyaYoneticisi()
        self._dosyadan_yukle()    # Başlangıçta dosyadan yükle
    def _dosyadan_yukle(self):
        # TODO: katalog.txt varsa kitapları yükle;hem kataloğa hem ağaca ekle.
        pass
    def kitap_ekle(self, isbn, ad, yazar, yil):
        self.katalog.kitap_ekle(isbn, ad, yazar, yil)
        self.agac.ekle(isbn, ad)
        self.gecmis.islem_kaydet('EKLE', f'{isbn} - {ad}')
        self.dosya.katalog_kaydet(self.katalog)
        self.dosya.gecmis_kaydet(f'EKLE|{isbn}|{ad}')
    def kitap_odunc_ver(self, isbn, ogrenci_no):
        # TODO: Kitap müsaitse ödünç ver, müsait değilse kuyruğa ekle.
        pass
    def kitap_iade_al(self, isbn):
        # TODO: Kitabı iade al; kuyrukta bekleyen varsa bildir.
        pass
    def kitap_ara(self, ad):
        # TODO: BST ile arama yap; sonucu döndür.
        pass
sistem = KutuphaneSistemi()
# 1. Kitap ekleme
sistem.kitap_ekle('978-006', 'Çalıkuşu',         'Reşat Nuri Güntekin', 1922)
sistem.kitap_ekle('978-007', 'Yaban',            'Yakup Kadri K.',      1932)
sistem.kitap_ekle('978-008', 'Dokuzuncu Hariciye Koğuşu', 'Peyami Safa', 1930)
sistem.kitap_ekle('978-009', 'Sinekli Bakkal',   'Halide Edib Adıvar',  1935)
sistem.kitap_ekle('978-010', 'Kuyucaklı Yusuf',  'Sabahattin Ali',      1937)
# 2. Katalog listeleme
sistem.katalog.kitaplari_listele()
# 3. BST ile arama
print(sistem.kitap_ara('Yaban'))
# 4. Ödünç verme
sistem.kitap_odunc_ver('978-003', 'Ogrenci_101')
sistem.kitap_odunc_ver('978-003', 'Ogrenci_102')  # Kuyruk testi
# 5. İade
sistem.kitap_iade_al('978-007')
# 6. İşlem geçmişi
sistem.gecmis.gecmisi_goster()
# 7. Geri al
sistem.gecmis.son_islemi_geri_al()
# 8. Alfabetik listeleme
for isbn, ad in sistem.agac.alfabetik_listele():
    print(f'{isbn} - {ad}')
```

6. Aşama: Projenizin modülerliğini ve okunurluğunu kontrol ediniz. Gerekli yerlerde açıklamalar kullanınız.

7. Aşama: Projenizin güncel kodlarını dijital deponuza gönderiniz. README.md dosyasını güncellemeyi unutmayınız.

Değerlendirme

a) Yapacağınız proje çalışması öğretmeniniz tarafından **Proje Derecelendirme Ölçeğinde** yer alan ölçütlere göre değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Verilen karekodu kullanarak derecelendirme ölçeğine ulaşabilirsiniz.

Proje derecelendirme ölçeğine ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

b) **Öz Değerlendirme Formu** proje geliştirme sürecinin sonunda kendinizi değerlendirmeniz amacıyla hazırlanmıştır. Cevaplarınızı tabloda ilgili kutucuğa yazınız. Verilen karekodu kullanarak öz değerlendirme formuna ulaşabilirsiniz.

Öz değerlendirme formuna ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



ÖLÇME VE DEĞERLENDİRME

A) Aşağıdaki yapılandırılmış gridde numaralandırılmış kutucuklarda temel veri yapıları ile ilgili bazı ifadeler verilmiştir. Kutucuk numaralarını kullanarak soruları cevaplayınız. Aynı kutucuk numarasını birden fazla cevap için kullanabilirsiniz.

(1) Yığın (Stack)	(2) Kuyruk (Queue)	(3) Bağlı Liste (Linked List)
(4) İkili Arama Ağacı (BST)	(5) AVL Ağacı	(6) LIFO
(7) FIFO	(8) Düğüm (Node)	(9) "a" (Append) Modu

1. Verilerin bellekte ardışık saklanma zorunluluğu olmayan, elemanların birbirine adres işaretçileriyle bağlandığı **doğrusal veri yapıları** hangileridir? (.....)

2. Bir web tarayıcısındaki **Geri** butonu ile bir metin editöründeki **Geri Al** mekanizmasının arkasında çalışan **veri yapısı** ve bu yapının esas aldığı **çalışma prensibi** hangi kutucuklarda verilmiştir? (.....)

3. Yazıcıya gönderilen belgelerin sıraya alınması, banka kuyrukları veya market kasasındaki işlem mantığını modellemek için kullanılan **veri yapısı** ve bu yapının temel **çalışma prensibi** hangileridir? (.....)

4. Gridde verilen yapılardan hangileri **hiyerarşik** bir organizasyon şemasına sahiptir? (.....)

5. Bellekte dinamik olarak büyüyen bağlı liste, yığın veya ağaç gibi yapıların her birinin içinde veri ve adres bilgisi barındıran **en temel ortak yapı** hangisidir? (.....)

6. Verilerin kalıcı depolama alanına yazılması sırasında, mevcut dosya içeriğine zarar vermeden ve eski kayıtları silmeden **dosyanın sonuna ekleme yapmayı** sağlayan dosya açma modu hangisidir? (.....)

7. Arama maliyetini azaltmayı hedefleyen, hiyerarşik yapıdaki arama ağaçlarından hangilerinde elemanlar belirli bir **sıralama düzenine veya yükseklik dengesine** göre yerleştirilir? (.....)

8. Bir müzik çalar uygulamasında hem **Sonraki Şarkı** tuşuyla ileriye gitmek hem de en son çalınan şarkılara dönebilmek için kullanılması gereken **veri yapısı** ve bu yapının kendi içinde ürettiği **temel bileşen** hangisidir? (.....)

B) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Bir bağlı liste yapısında araya yeni bir düğüm eklemek için aşağıdaki işlemlerden hangisinin yapılması yeterlidir?

A) Tüm listeyi yeni baştan oluşturmak
B) Bellekteki tüm verileri birer hücre kaydırmak
C) İlgili düğümlerin işaretçilerini (pointer) yeni düğümü gösterecek şekilde güncellemek
D) Listeyi önce bir diziye (array) dönüştürüp sonra ekleme yapmak
E) Verileri alfabetik olarak yeniden sıralamak

2. Bir metin düzenleyicide yapılan işlemlerin "Geri Al" tuşuyla sondan başa doğru iptal edilmesi, hangi veri yapısının çalışma prensibine bir örnektir?

A) Kuyruk
B) Yığın
C) İkili Arama Ağacı
D) Dairesel Bağlı Liste
E) AVL Ağacı

3. İşletim sistemlerinde yazıcıya gönderilen belgelerin, gönderilme sırasına göre yazdırılmasını sağlayan veri yapısı ve prensibi aşağıdakilerden hangisidir?

A) Yığın
B) Bağlı Liste
C) Kuyruk
D) AVL Ağacı
E) İkili Arama Ağacı

4. İkili Arama Ağacı yapısında bir veri aranırken, her adımda aranan değerin kök düğümden büyük veya küçük olmasına bakılarak dallardan birine gidilir. Bu durum arama hızını hangi seviyeye indirir?

A) O(n)
B) O(1)
C) $O(n^2)$
D) O(log n)
E) O(n!)

5. Bir bağlı listede son düğümün işaretçisi None yerine listenin başındaki düğümü gösteriyorsa bu özel yapı aşağıdakilerden hangisidir?

A) Tek Yönlü Bağlı Liste
B) Çift Yönlü Bağlı Liste
C) Yığın
D) Kuyruk
E) Dairesel Bağlı Liste

C) 1-3. soruları aşağıdaki metne göre cevaplayınız.

Yazılım dünyasında verileri organize etmek için farklı yapılar kullanılır. Örneğin bir web tarayıcısında "Geri" butonuna basıldığında yapılan işlemleri geri almak için Kuyruk (Queue) veri yapısı kullanılır. Çünkü bu yapı, son yapılan işlemin en üstte tutulduğu LIFO prensibiyle çalışır. Öte yandan, bir müzik çalar listesinde şarkılar arasında ileri ve geri gidilebilmesi için Tek Yönlü Bağlı Liste yapısı en ideal çözümdür.

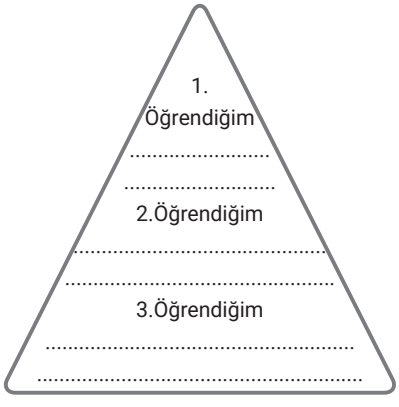
Hiyerarşik veriler söz konusu olduğunda ise ağaç yapıları devreye girer. İkili Arama Ağacı (BST), arama işlemlerini hızlandırmak için verileri rastgele dallara yerleştirir. Bu ağaçların dengesizleşerek bir bağlı listeye dönüşmesini engellemek için AVL Ağacı gibi kendi kendini dengeleyen yapılar geliştirilmiştir. Program kapansa bile verilerin saklanması için dosya işlemleri yapılır. Bir dosyanın mevcut içeriğini koruyup üzerine yeni veri eklemek için dosya "**w**" (write) modunda açılmalıdır.

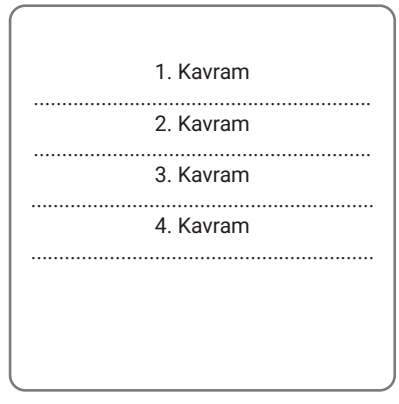
1. Metinde geçen beş hatalı bilgiyi bularak doğrularını yazınız.

2. Müzik çalar uygulamasında şarkılar arasında ileri ve geri gidilebilmesi için metinde önerilen yapı yerine hangi bağlı liste türü kullanılmalıdır? Neden?

3. Bir öğrenci listesine yeni bir isim eklenmek istendiğinde mevcut isimlerin silinmemesi için hangi dosya açma modunu tercih edilmesi gerekir? Neden?

Ç) Temel veri yapıları hakkında öğrendiğiniz üç bilgiyi üçgenin, bu öğrenme biriminde ilk defa karşılaştığınız dört temel kavramı karenin, aklınıza takılan bir soruyu çemberin içine yazınız.







KAYNAKÇA

Çalışma ve Sosyal Güvenlik Bakanlığı. (2012). İş sağlığı ve güvenliği risk değerlendirmesi yönetmeliği.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. ve Stein, C. (2022). Algoritmalara giriş (4. bs., H. Yıldız, Çev.). Ankara: Nobel Akademik Yayıncılık.

International Labour Office (ILO). (2013). National system for recording and notification of occupational diseases: Practical guide. International Labour Organization.

Lutz, M. (2013). Learning Python (5. bs.). Sebastopol, CA: O'Reilly Media.

Occupational Safety and Health Administration (OSHA). (2016). Recommended practices for safety and health programs. U.S. Department of Labor.

"Python Software Foundation. (2024). The Python language reference (Sürüm 3.12). Erişim tarihi: 09 Haziran 2026, https://docs.python.org/3/reference/

Python Software Foundation. (2024). The Python tutorial (Sürüm 3.12). Erişim tarihi: 09 Haziran 2026, https://docs.python.org/3/tutorial/

T.C. Millî Eğitim Bakanlığı. (2026). Yapay zekâ alanı çerçeve öğretim programı. Mesleki ve Teknik Eğitim Genel Müdürlüğü.

Türkiye Cumhuriyeti. (2012). 6331 Sayılı İş Sağlığı ve Güvenliği Kanunu.

World Health Organization (WHO). (2022). WHO guidelines on mental health at work. World Health Organization.

APA 7 kaynakça düzeni esas alınarak hazırlanmıştır.

GENEL AĞ, GÖRSEL, E-İÇERİK KAYNAKÇASI

Kitabın görsel kaynakçası, genel ağ kaynakçası ve e-içerik kaynakçasına bu karekodu okutarak ulaşabilirsiniz.

Dijital dokümanlarda karekodun üzerine tıklayarak içeriğe ulaşabilirsiniz.

CEVAP ANAHTARLARI

Cevap anahtarlarına bu karekodu okutarak ulaşabilirsiniz.

Dijital dokümanlarda karekodun üzerine tıklayarak içeriğe ulaşabilirsiniz.


ROMANYA
Bükreş
UKRAYNA
RUSYA FEDERASYONU
HAZAR DENİZİ
Sofya
BULGARİSTAN
K A R A D E N İ Z
GÜRCİSTAN
Tiflis
Edirne
Kırklareli
Sinop
Bartın
Zonguldak
Kastamonu
ACARA Ö.C.
Artvin
Ardahan
ERMENİSTAN
AZERBAYCAN
Kura N.
YUNANİSTAN
Tekirdağ
İstanbul
Samsun
Ordu
Giresun
Trabzon
Rize
Kars
Erivan
Meriç N.
MARMARA DENİZİ
Kocaeli
Düzce
Karabük
Yalova
Sakarya
Bolu
Çankırı
Amasya
Çoruh N.
Gümüşhane
Bayburt
Gökçeada
Çanakkale
Bursa
Bilecik
Sakarya N.
Çorum
Tokat
Yeşilırmak N.
Kızılırmak N.
Ankara
Erzincan
Erzurum
Iğdır
Aras N.
NAHÇIVAN Ö.C. (AZERBAYCAN)
Balıkesir
Eskişehir
Kırıkkale
Yozgat
Sivas
Ağrı
Bakırçay N.
Kütahya
Kırşehir
Tunceli
Bingöl
Muş
Manisa
Gediz N.
Afyonkarahisar
Tuz Gölü
Elazığ
Van Gölü
Van
Atina
İzmir
Uşak
Nevşehir
Kayseri
Bitlis
Küçük Menderes N.
Aksaray
Malatya
Urumiye Gölü
Aydın
Büyük Menderes N.
Konya
Diyarbakır
Siirt
Batman
Hakkâri
Denizli
Burdur
Isparta
Niğde
Adıyaman
Dicle N.
Şırnak
Kahramanmaraş
Muğla
Mardin
İ R A N
Karaman
Osmaniye
Gaziantep
Şanlıurfa
Antalya
Adana
Mersin
Kilis
Göksu N.
Seyhan N.
Ceyhan N.
Hatay
Fırat N.
Asi N.
KUZEY KIBRIS TÜRK CUMHURİYETİ
Erenköy
Lefkoşa
S U R İ Y E
I R A K
GÜNEY KIBRIS RUM YÖNETİMİ
LÜBNAN
Beyrut
Bağdat
Şam
A K D E N İ Z
L İ B Y A
Tel Aviv
FİLİSTİN
Batı Şeria
Doğu Kudüs
FİLİSTİN
Gazze Şeridi
İSRAİL
M I S I R
Nil N.
Süveyş K.
EGE DENİZİ
TÜRKİYE'NİN MÜLKİ İDARE BÖLÜMLERİ İLE KARA VE DENİZ KOMŞULARI HARİTASI
BAŞKENT
Ankara
İL MERKEZİ
Elazığ
GÖL
ÜLKE SINIRI
ÖZERK BÖLGE SINIRI
AKARSU
0 50 100 200 300 400
Km


TÜRK DÜNYASI HARİTASI
Bağımsız Türk Devletleri
Türk Cumhuriyetleri, Türk Bölgeleri
Türk Toplulukları
0 250 500 1,000 1,500 2,000 Km
TÜRK CUMHURİYETLERİ, TÜRK BÖLGELERİ
1 GAGAVUZ YERİ
2 KARAÇAY-ÇERKESYA
3 KABARDİN-BALKARYA C.
4 DAĞISTAN C.
5 ÇUVAŞİSTAN C.
6 TATARİSTAN C.
7 BAŞKURT C.
8 KARAKALPAKİSTAN C.
9 ALTAY C.
10 HAKASYA C.
TÜRK TOPLULUKLARI
a Avrupa'daki Türkler
b Kosova Türkleri
c Yunanistan Türkleri
ç Bulgaristan Türkleri
d Romanya Türkleri
e Suriye Türkmenleri
f Irak Türkmenleri
g İran Türkleri
h Afganistan Türkmenleri
ı Pakistan Türkmenleri
i Hindistan Türkmenleri
j Çin H. C. Türkleri
k Moğolistan Türkleri
l Rusya Federasyonu Türkleri
ÜLKE KISALTMALARI
ARNAVUTLUK ARN
BELÇİKA BEL
BOSNA - HERSEK BH
GÜNEY KIBRIS RUM YÖNETİMİ GKRY
HIRVATİSTAN HRV
İSVİÇRE İS
KARADAĞ KD
KOSOVA KS
KUZEY KIBRIS TÜRK CUMHURİYETİ KKTC
KUZEY MAKEDONYA KMKD
LİHTENŞTAYN LİH
SIRBİSTAN SR
RUSYA FEDERASYONU
KAZAKİSTAN
TÜRKİYE
İRAN
AFGANİSTAN
PAKİSTAN
HİNDİSTAN
MOĞOLİSTAN
ÇİN HALK CUMHURİYETİ
SAHA (YAKUT) C.
TUVA C.
SİNCAN UYGUR ÖZERK BÖLGESİ
Norveç Denizi
Barents Denizi
Kara Denizi
Ohotsk Denizi
Bering Denizi
Kuzey Denizi
Akdeniz
Karadeniz
Hazar Denizi
Umman Denizi
Yemen Denizi
Kızıldeniz
Moskova
Astana
Bişkek
Taşkent
Aşkabat
Dışanbe
Kabil
İslamabad
Yeni Delhi
Dakka
Pekin
Ulanbator
Ankara
Bakü
Tahran
Bağdat
Riyad
Doha
Muskat
Sana
Amman
Kahire
Trablus
Tunus
Cezayir
Rabat
Madrid
Paris
Berlin
Varşova
Kiev
Minsk
Helsinki
Stockholm
Oslo
Londra
Dublin
Brüksel
Amsterdam
Kopenhag
Viyana
Prag
Budapeşte
Bükreş
Sofya
Atina
Belgrad
Saraybosna
Zagreb
Lefkoşa
Şam
Beyrut
Kudüs
Amman
Abu Dabi
Kuveyt
Nijer
Çad
Sudan
Libya
Mısır
Fas
Portekiz
İspanya
Fransa
Almanya
Polonya
Belarus
Ukrayna
Moldova
İtalya
Yunanistan
Bulgaristan
Romanya
Macaristan
Avusturya
İsviçre
Hollanda
Belçika
Danimarka
Norveç
İsveç
Finlandiya
İrlanda
İzlanda
Cebelitarık B.
Manş Denizi
Tiren Denizi
Adriyatik Denizi
Ege Denizi
Baltık D.
Ural N.
Volga N.
Don N.
Dinyeper N.
Ob N.
İrtiş N.
Lena N.
Yenisey N.
Amur N.
Nil N.
Fırat N.
Dicle N.
Sırderya N.
Amuderya N.
Balkaş G.
Aral G.
Baykal G.
Hotan N.
Sarı N.
Ganj N.
Hint Okyanusu
Taraksız Bölge
CUMHURİYET YEKŞİR
Ürümçi
Lhasa
Kaşgar
Hotan
Kuşadası
Kıbrıs
Girit
Sicilya
Sardunya
Korsika
Malta
Tunus
Cezayir
Fas
Trablus
Libya
Nijer
Çad
Sudan
Mısır
Filistin
Irak
Kuveyt
Suudi Arabistan
Katar
Birleşik Arap Emirlikleri
Umman
Yemen
Pakistan
Hindistan
Bangladeş
İran
Afganistan
Türkmenistan
Özbekistan
Kırgızistan
Tacikistan
Kazakistan
Rusya Federasyonu
Moğolistan
Çin Halk Cumhuriyeti
Türkiye
Gürcistan
Azerbaycan
Ermenistan
Kıbrıs
Suriye
Irak
İran
Afganistan
Pakistan
Hindistan
Bangladeş
Çin H. C.
Moğolistan
Rusya Federasyonu