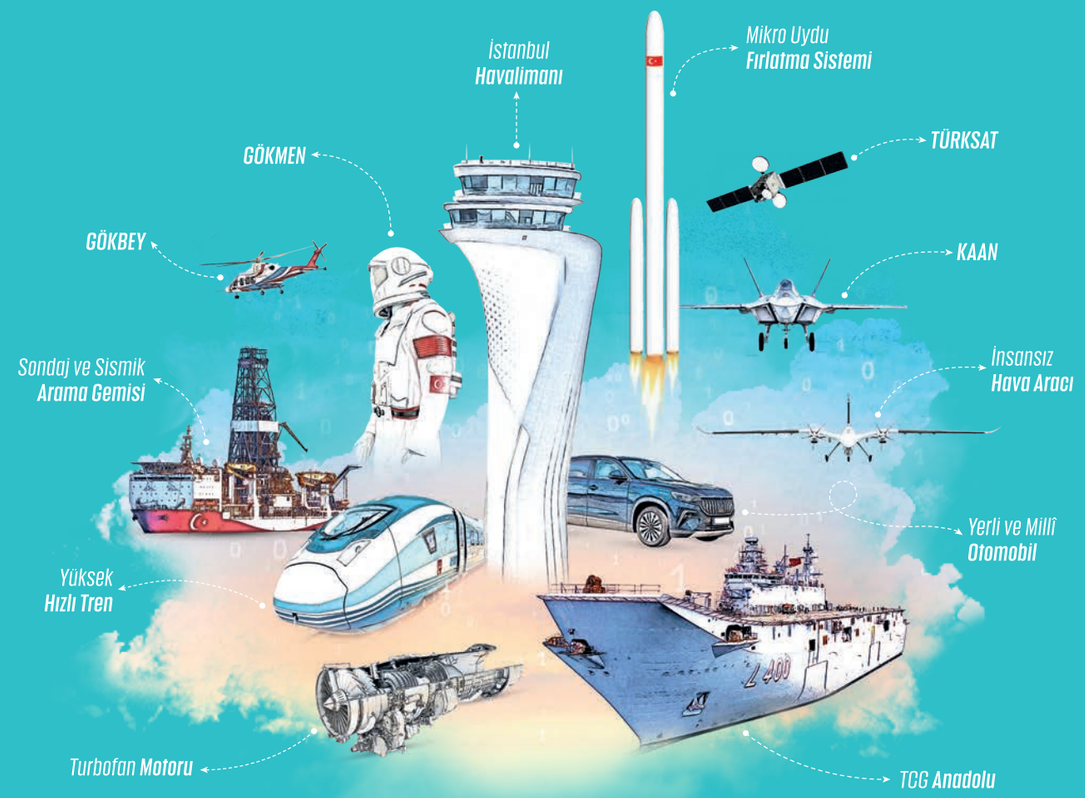
Bu kitaba sığmayan daha neler var!

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: 978-625-5498-37-3

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

BİLİŞİM TEKNOLOJİLERİ ALANI

PROGRAMLAMA TEMELLERİ 9

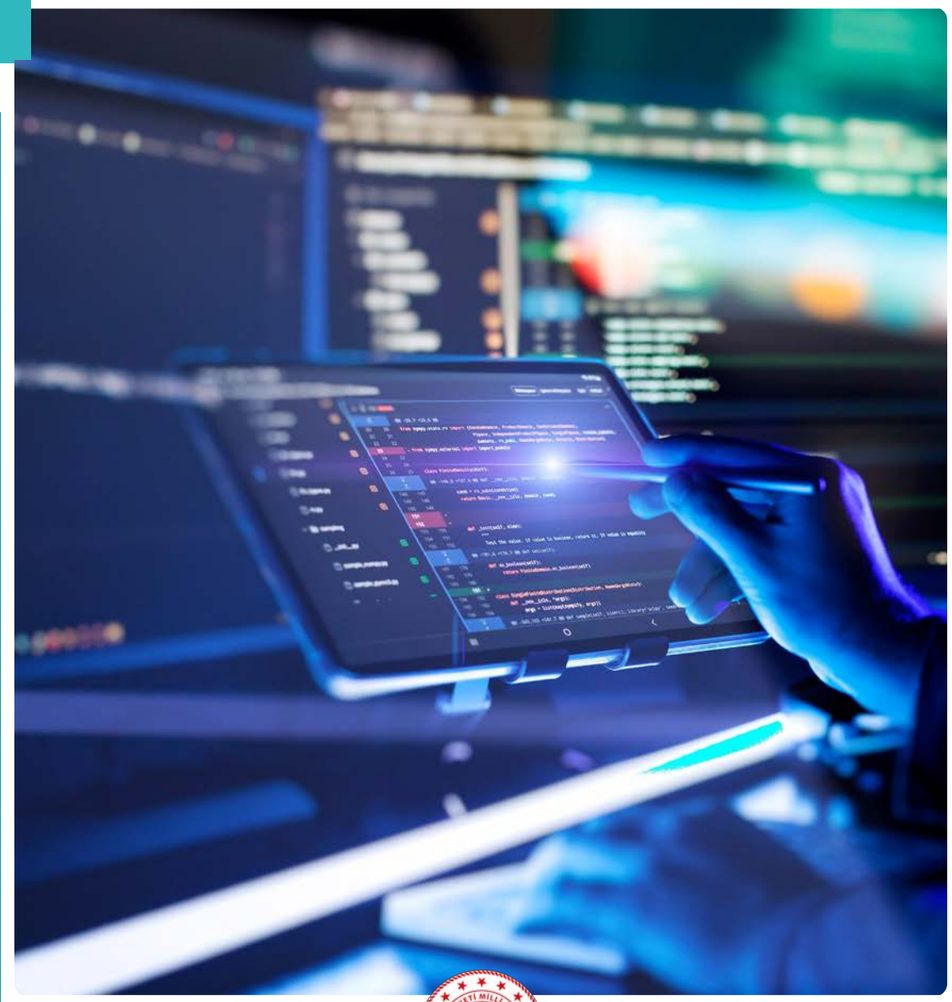
DERS MATERYALİ

MESLEKİ VE TEKNİK ANADOLU LİSESİ

BİLİŞİM TEKNOLOJİLERİ ALANI

9 DERS MATERYALİ

PROGRAMLAMA TEMELLERİ



MESLEKİ VE TEKNİK ANADOLU LİSESİ

BİLİŞİM TEKNOLOJİLERİ ALANI

PROGRAMLAMA TEMELLERİ

9

DERS MATERYALİ

Yazarlar

Ebru YAĞMUR
Mustafa CAYMAZ
Tuncay BİLDİK

MİLLÎ EĞİTİM BAKANLIĞI YAYINLARI .....: 10942
YARDIMCI VE KAYNAK KİTAPLAR DİZİSİ .....: 3861

Her hakkı saklıdır ve Millî Eğitim Bakanlığına aittir.
Ders materyalinin metin, soru ve şekilleri kısmen de olsa hiçbir surette alınıp yayımlanamaz.

HAZIRLAYANLAR

Dil Uzmanı
Merve SERBEST ERTOĞAN

Program Geliştirme Uzmanı
Pelin BÖLÜKBAŞI

Rehberlik Uzmanı
Davut ŞENYÜREK

Ölçme ve Değerlendirme Uzmanı
Gülhan ŞAHİN

Görsel Tasarım Uzmanı
Özden ALTUN

ISBN: 978-625-5498-37-3

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

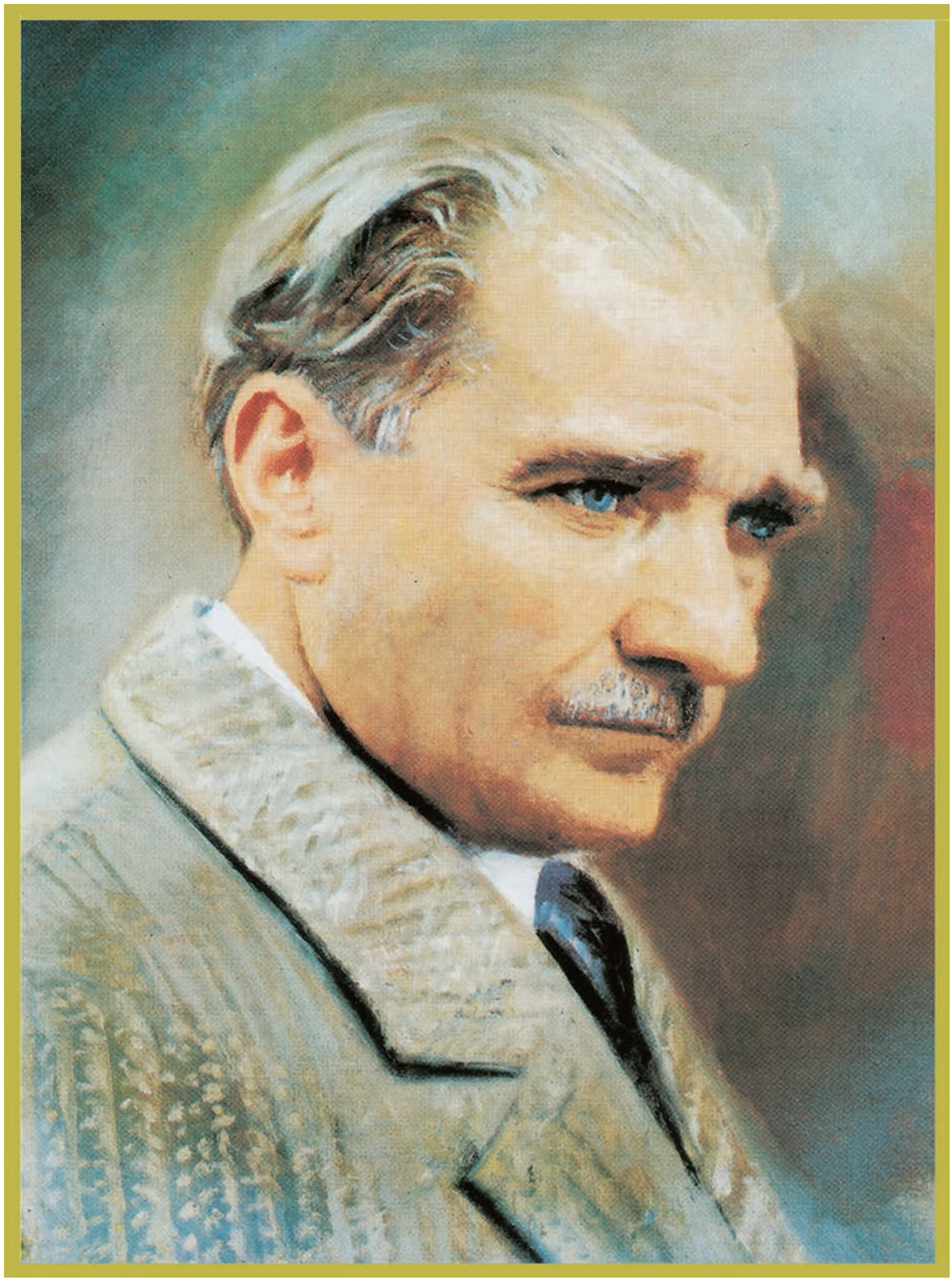
GENÇLİĞE HİTABE

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

İÇİNDEKİLER

DERS MATERYALİNİN TANITIMI 13

1. ÖĞRENME BİRİMİ: ALGORİTMİK DÜŞÜNME VE TASARIM 17



1.1. PROBLEM ANALİZİ VE BİLEŞENLERİ 19

1.1.1. Algoritma ve Tasarım Kriterleri 19

1.1.2. Problemi Bileşenlerine Ayırma 21

1.1.3. Girdi-İşlem-Çıktı Modeli 21

1.2. AKIŞ ŞEMASI (FLOWCHART) 24

1.2.1. Sözde Kod ve Elle Çalıştırma 24

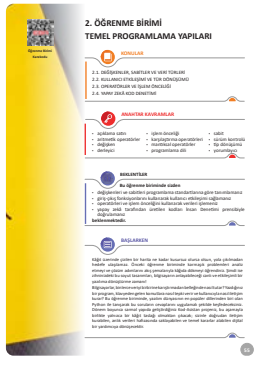
1.2.2. Akış Şeması Sembolleri ve Görsel Modelleme 30

1.2.3. Algoritmalarda Metinsel Veriler ve İşlemler 41

1.2.4. Dijital Depo 42

ÖLÇME VE DEĞERLENDİRME 48

2. ÖĞRENME BİRİMİ: TEMEL PROGRAMLAMA YAPILARI 57



2.1. DEĞİŞKENLER, SABİTLER VE VERİ TÜRLERİ 57

2.1.1. Programlama Dillerinin Seviyeleri 57

2.1.2. Derleyiciler ve Yorumlayıcılar 58

2.1.3. Python Dili 59

2.1.4. Python Interactive Shell ve Idle 61

2.1.5. Yazılım Geliştirme Ortamları 63

2.1.6. Temel Veri Türleri 68

2.1.7. type() Fonksiyonuyla Veri Türü Analizi 69

2.1.8. Değişken Tanımlama Kuralları 70

2.1.9. Değişken Adlandırma Standartları (Notasyonlar) 70

2.1.10. Sabitler 72

2.2. KULLANICI ETKİLEŞİMİ VE TÜR DÖNÜŞÜMÜ 75

2.2.1. input() ve print() Fonksiyonları 75

2.2.2. Temel Tür Dönüşüm Fonksiyonları ve Sınır Durum Analizi 82

2.3. OPERATÖRLER VE İŞLEM ÖNCELİĞİ 83

2.3.1. Operatörler 83

2.3.2. İşlem Önceliği Kuralları 88

2.4. YAPAY ZEKÂ KOD DENETİMİ 90

2.4.1. İnsan Denetimi İlkesi 90

2.4.2. Yapay Zekâ Destekli Yazılım Geliştirmede Sorumluluk Sınırı 90

ÖLÇME VE DEĞERLENDİRME 93



3. ÖĞRENME BİRİMİ: KARAR YAPILARI VE DÖNGÜLER 103

3.1. KOŞUL YAPILARI VE MANTIKSAL OPERATÖRLER 103

3.1.1. Tek Yönlü Karar Mekanizmaları: if (Eğer) Yapısı 103

3.1.2. Dallanma Mantığı ve Girinti Dünyası 103

3.1.3. Çift Yönlü Alternatifler: if-else (Eğer-Değilse) Yapısı 103

3.1.4. Karşılaştırma ve Mantıksal Operatörlerin Birleşik Kullanımı 104

3.1.5. Çok Kriterli Karar Mekanizmaları: if-elif-else Merdiveni 106

3.1.6. İç İçe Koşul Yapıları 106

3.1.7. Kararlı Veri Kontrolü: Büyük / Küçük Harf Duyarlılığı 111

3.2. DÖNGÜ YAPILARI VE LİSTE İŞLEMLERİ 112

3.2.1. Belirli Sayıda Tekrar: for Döngüsü ve Sıralı Veri Dünyası 112

3.2.2. range() Fonksiyonu 114

3.2.3. Koşula Bağlı Tekrar: while Döngüsü ve Mantıksal Süreklilik 114

3.2.4. Bileşik Atama Operatörleri 114

3.2.5. Döngü Kontrol İfadeleri: break ve continue 116

3.2.6. Döngülerin Yazılım Verimliliğindeki Rolü 120

3.2.7. for ve while Döngüleri Arasındaki Karşılaştırmalı Farklar 120

3.2.8. Liste İşlemleri (Veri Kümeleri ve Döngü Entegrasyonu) 123

3.2.9. Dinamik Döngü-Liste İlişkisi (Iterating Through a List) 124

3.2.10. Liste Metotları ve Bellek Davranışları 128

3.2.11. Döngü ve Liste İşlemlerinde Yapay Zekâ Denetimi 130

3.3. GRAFİKSEL GÖRSELLEŞTİRME 133

3.3.1. Koordinat Tabanlı Çizim Aracı: Turtle 133

3.3.2. İç İçe Döngülerle Karmaşık Geometrik Desenlerin Üretimi 137

3.3.3. Dijital Depolarda Sürüm Yönetimi ve Release Kavramı 140

ÖLÇME VE DEĞERLENDİRME 144



4. ÖĞRENME BİRİMİ: MODÜLER PROGRAMLAMA VE FONKSİYONLAR 152

4.1. FONKSİYON TANIMLAMA VE VERİ AKIŞI 152

4.1.1. Kullanıcı Tanımlı Fonksiyonlar 154

4.1.2. Fonksiyon Tanımlama Standartları (def Anahtar Kelimesi)..... 154

4.1.3. Parametre Yönetimi ve Veri Aktarımı (Girdi) 156

4.1.4. Dönüş Değeri İfadesinin Veri Aktarma Rolü (Çıktı) 156

4.1.5. Argüman Aktarım Yöntemleri ve Pozisyonel Eşleşme..... 161

4.1.6. Veri Akışında Esneklik: Varsayılan Değerli Parametreler 162

4.2. MODÜLER MİMARİ VE DEĞİŞKEN ETKİ ALANI..... 166

4.2.1. Bellekte İsim Çakışmaları ve Gizleme Kuralları 166

4.2.2. Kodun Fiziksel Parçalanması: Modül Oluşturma ve İçe Aktarma 169

4.2.3. Veri Akışında Esneklik: Modüllere ve Fonksiyonlara Takma İsim Vermek 172

4.3. KOD KALİTESİ VE DENETİMİ..... 176

4.3.1. Yazılım Standartları ve Teknik Dokümantasyon 176

4.3.2. Kod Yeniden Yapılandırma 177

ÖLÇME VE DEĞERLENDİRME 183



5. ÖĞRENME BİRİMİ: VERİ YÖNETİMİ VE KALICI PROGRAM YAPILARI..... 190

5.1. SÖZLÜK VERİ YAPISI (DICTIONARY)..... 190

5.1.1. Sözlük Elemanlarına Erişim ve Güvenli Veri Erişimi 191

5.1.2. Sözlüklerde Veri Ekleme ve Güncelleme 193

5.1.3. Sözlüklerden Veri Silme 194

5.1.4. Sözlük İçeriğini Toplu Görüntüleme ve Analiz Metotları 195

5.1.5. İç İçe Yapılarla Veri Modelleme: Listeler ve Sözlükler..... 198

5.2. DOSYA İŞLEMLERİ (I/O) VE VERİ KALICILIĞI..... 204

5.2.1. Dosya İşlemlerinde Temel Mantık ve open() Fonksiyonu 204

5.2.2. Dosya Yolu (Path) Yönetimi: Mutlak ve Göreceli Yollar 204

5.2.3. with open() Bağlam Yöneticisi 205

5.2.4. Dosyadan Veri Okuma Yöntemleri..... 205

5.3. VERİ FORMATLARI VE DENETİM 210

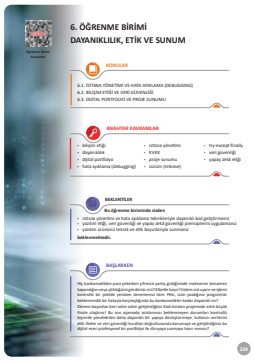
5.3.1. Sözlük Yapılarını JSON Formatında Saklama ve Okuma..... 210

5.3.2. Kod Editörü ile Yerel Sürüm Kontrolü (Git) ve GitHub Entegrasyonu 212

5.3.3. Veri Bütünlüğü ve Sürüm Kontrolü: .gitignore Yapılandırması 216

5.3.4. Yapay Zekâ ve Veri Formatı Denetimi 217

ÖLÇME VE DEĞERLENDİRME 223



6. ÖĞRENME BİRİMİ: DAYANIKLILIK, ETİK VE SUNUM 230

6.1. İSTİSNA YÖNETİMİ VE HATA AYIKLAMA (DEBUGGING) 230

6.1.1. Try-Except Yapısı 232

6.1.2. Traceback Mesajları ve Hata Ayıklama (Debugging) 241

6.2. BİLİŞİM ETİĞİ VE VERİ GÜVENLİĞİ 243

6.2.1. Veri Güvenliği ve KVKK 243

6.2.2. Yapay Zekâ Etiği 243

6.2.3. Kullanıcı Girdilerinin Doğrulanması ve Arındırılması 244

6.3. DİJİTAL PORTFOLYO VE PROJE SUNUMU 249

6.3.1. Dijital Portfolyo ve Teknik Dokümantasyon (README.md) 249

6.3.2. Final Sürümünün Yayımlanması 250

ÖLÇME VE DEĞERLENDİRME 254

KAYNAKÇA 259

CEVAP ANAHTARI 260

DERS MATERYALİNİN TANITIMI

Öğrenme birimi karekodunu gösterir.

Öğrenme birimi adını gösterir.

Öğrenme birimi numarasını gösterir.

Öğrenme birimi konu başlıklarını gösterir.



Öğrenme birimine uygun görseli içerir.

Öğrenme biriminde yer alan temel kavramları gösterir.

Öğrenme birimindeki beklentileri gösterir.

MATERYAL GENELİNDE KULLANILAN İKONLAR

KONULAR ANAHTAR KAVRAMLAR BEKLENTİLER BAŞLARKEN

SIRA SİZDE 1. UYGULAMA 1. ÖRNEK

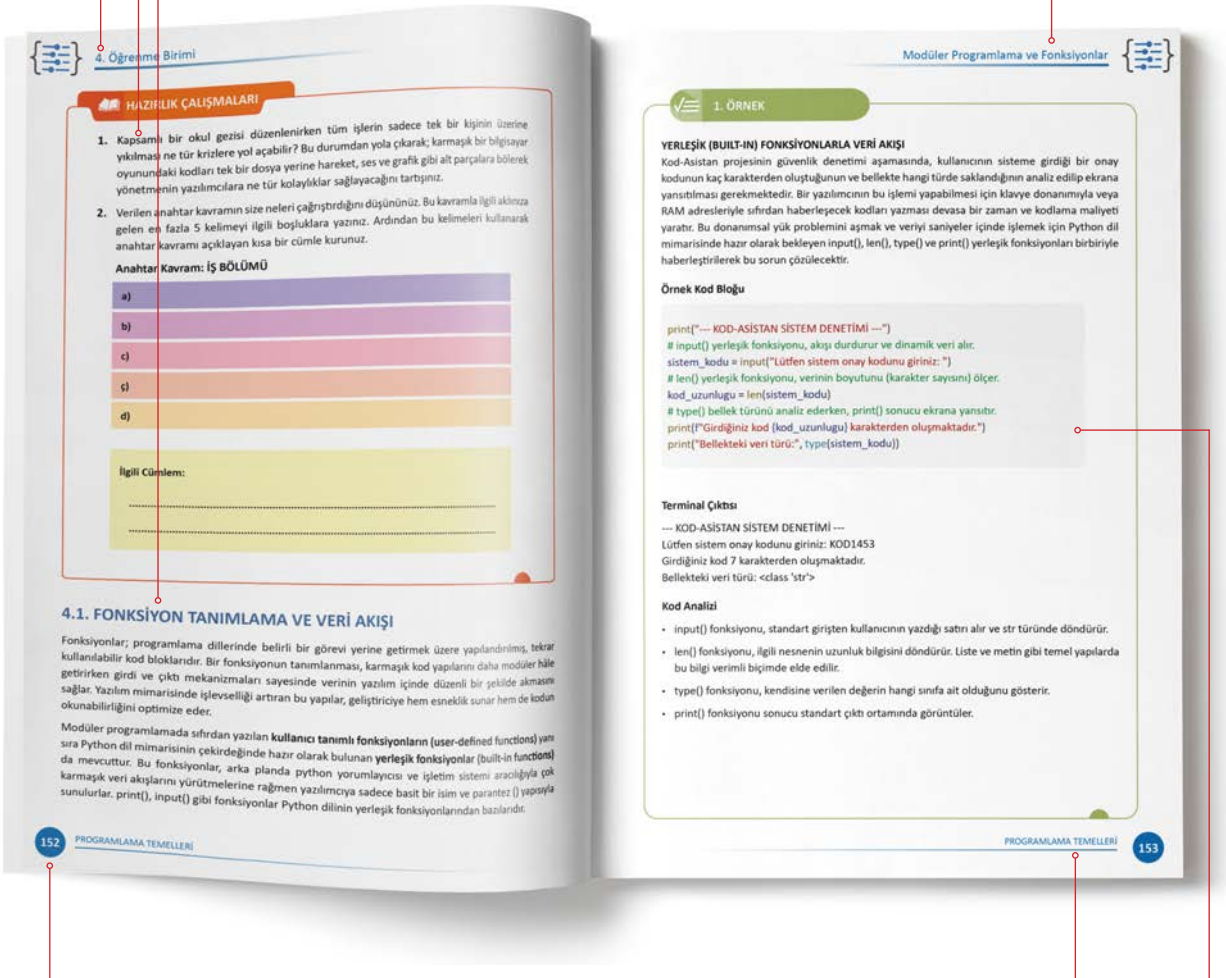
DERS MATERYALİNİN TANITIMI

Öğrenme birimi numarasını gösterir.

Öğrenme birimine yönelik ön bilgilerin yoklanması adına sorulan soruları içerir.

Öğrenme birimi adını gösterir.

Konu başlıklarını gösterir.



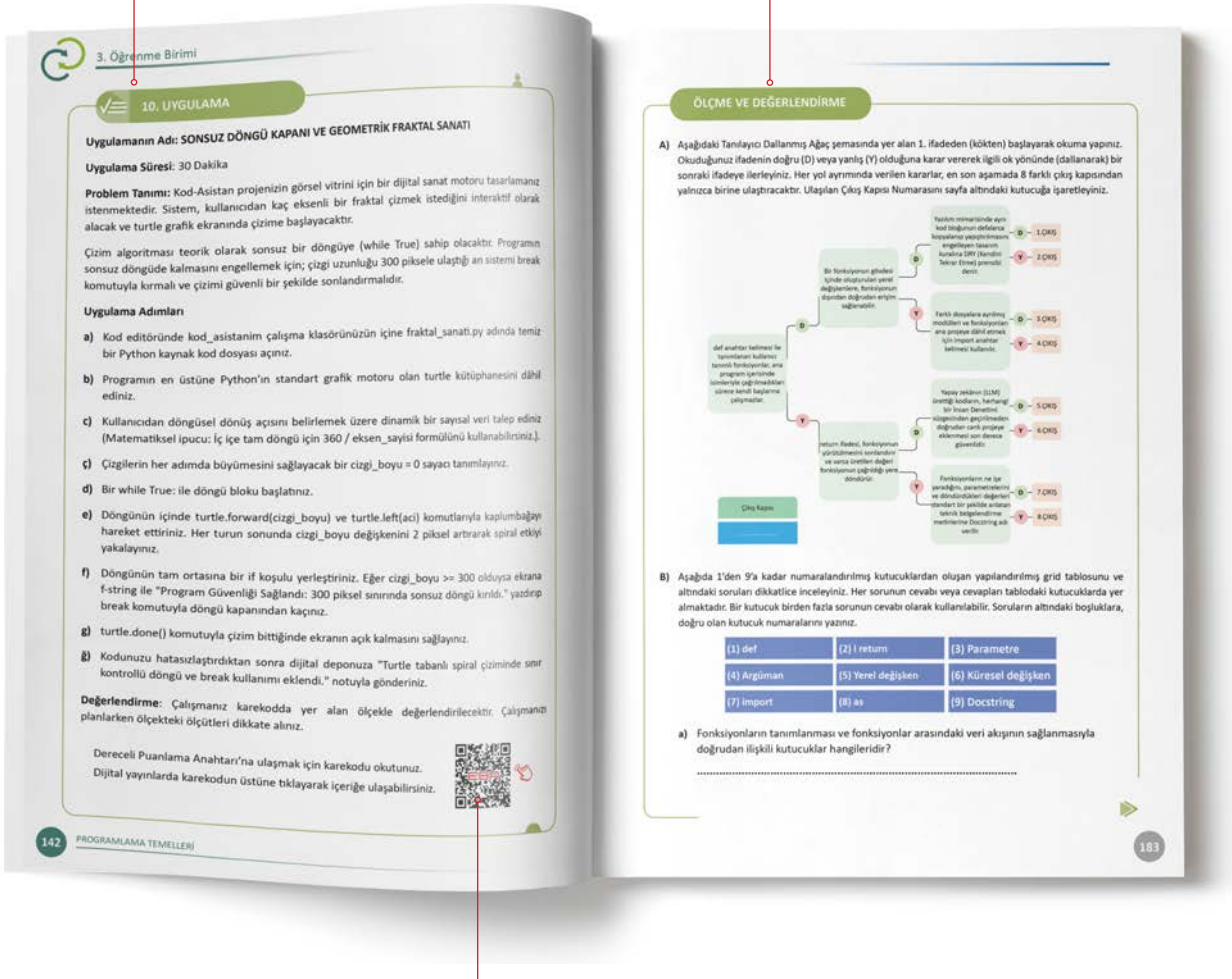
Sayfa numarasını gösterir.

Öğrenme birimindeki örnek kod blokunu gösterir.

Materyal adını gösterir.

Öğrenme biriminde yer alan uygulamaları gösterir.

Öğrenme birimindeki ölçme ve değerlendirmeyi gösterir.



3. Öğrenme Birimi

10. UYGULAMA

Uygulamanın Adı: SONSUZ DÖNGÜ KAPANI VE GEOMETRİK FRAKTAL SANATI

Uygulama Süresi: 30 Dakika

Problem Tanımı: Kod-Asistan projenizin görsel vitrini için bir dijital sanat motoru tasarlamanız istenmektedir. Sistem, kullanıcıdan kaç eksenli bir fraktal çizmek istediğini interaktif olarak alacak ve turtle grafik ekranında çizime başlayacaktır.

Çizim algoritması teorik olarak sonsuz bir döngüye (while True) sahip olacaktır. Programın sonsuz döngüde kalmasını engellemek için; çizgi uzunluğu 300 piksele ulaştığı an sistemi break komutuyla kırmalı ve çizimi güvenli bir şekilde sonlandırmalıdır.

Uygulama Adımları

a) Kod editöründe kod_asistanim çalışma klasörünüzün içine fraktal_sanati.py adında temiz bir Python kaynak kod dosyası açınız.

b) Programın en üstüne Python'ın standart grafik motoru olan turtle kütüphanesini dâhil ediniz.

c) Kullanıcıdan döngüsel dönüş açısını belirlemek üzere dinamik bir sayısal veri talep ediniz (Matematiksel ipucu: İç içe tam döngü için 360 / eksen_sayisi formülünü kullanabilirsiniz.).

ç) Çizgilerin her adımda büyümesini sağlayacak bir cizgi_boyu = 0 sayacı tanımlayınız.

d) Bir while True: ile döngü bloku başlatınız.

e) Döngünün içinde turtle.forward(cizgi_boyu) ve turtle.left(aci) komutlarıyla kaplumbağayı hareket ettiriniz. Her turun sonunda cizgi_boyu değişkenini 2 piksel artırarak spiral etkiyi yakalayınız.

f) Döngünün tam ortasına bir if koşulu yerleştiriniz. Eğer cizgi_boyu >= 300 olduysa ekrana f-string ile "Program Güvenliği Sağlandı: 300 piksel sınırında sonsuz döngü kırıldı." yazdırıp break komutuyla döngü kapanından kaçınız.

g) turtle.done() komutuyla çizim bittiğinde ekranın açık kalmasını sağlayınız.

ğ) Kodunuzu hatasızlaştırdıktan sonra dijital deponuza "Turtle tabanlı spiral çiziminde sınır kontrollü döngü ve break kullanımı eklendi." notuyla gönderiniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

142 PROGRAMLAMA TEMELLERİ

ÖLÇME VE DEĞERLENDİRME

A) Aşağıdaki Tanılayıcı Dallanmış Ağaç şemasında yer alan 1. ifadeden (kökten) başlayarak okuma yapınız. Okuduğunuz ifadenin doğru (D) veya yanlış (Y) olduğuna karar vererek ilgili ok yönünde (dallanarak) bir sonraki ifadeye ilerleyiniz. Her yol ayrımında verilen kararlar, en son aşamada 8 farklı çıkış kapısından yalnızca birine ulaştıracaktır. Ulaşılan Çıkış Kapısı Numarasını sayfa altındaki kutucuğa işaretleyiniz.

B) Aşağıda 1'den 9'a kadar numaralandırılmış kutucuklardan oluşan yapılandırılmış grid tablosunu ve altındaki soruları dikkatlice inceleyiniz. Her sorunun cevabı veya cevapları tablodaki kutucuklarda yer almaktadır. Bir kutucuk birden fazla sorunun cevabı olarak kullanılabilir. Soruların altındaki boşluklara, doğru olan kutucuk numaralarını yazınız.

(1) def	(2) I return	(3) Parametre
(4) Argüman	(5) Yerel değişken	(6) Küresel değişken
(7) import	(8) as	(9) Docstring

a) Fonksiyonların tanımlanması ve fonksiyonlar arasındaki veri akışının sağlanmasıyla doğrudan ilişkili kutucuklar hangileridir?

.....

183

Öğrenme birimindeki uygulama sonrasında yapılacak değerlendirmelere ait karekodları gösterir.





Öğrenme Birimi Karekodu

1. ÖĞRENME BİRİMİ
ALGORİTMİK DÜŞÜNME VE TASARIM

KONULAR

1.1. PROBLEM ANALİZİ VE BİLEŞENLERİ

1.2. AKIŞ ŞEMASI (FLOWCHART)

ANAHTAR KAVRAMLAR

- akış şeması
- algoritma
- çıktı
- dijital depo
- girdi
- işlem
- karşılaştırma operatörleri
- sözde kod

BEKLENTİLER

Bu öğrenme biriminde sizden

- algoritmik düşünme sürecini problem senaryoları üzerinde yapılandırmanız
- çözüm algoritmaları için akış şemaları hazırlamanız

beklenmektedir.

BAŞLARKEN

Hiç gözleri kapalı bir arkadaşınıza odanın diğer ucundaki bir eşyayı bulması için adım adım tarif vermeyi denediniz mi? "Sağa dön, üç adım at, elini uzat..." Eğer tek bir detayı bile atlarsanız arkadaşınız muhtemelen duvara çarpacaktır! İşte bilgisayarlar da tam olarak böyle çalışır; kendilerine tanımlanan komutlar ve kurallar doğrultusunda işlem yaparlar ve ürettikleri sonuçların doğruluğu büyük ölçüde kullandığınız algoritmanın ve verilerin doğruluğuna bağlıdır.

Bu öğrenme biriminde, zihninizdeki problem çözme yöntemlerini bilgisayarın anlayabileceği sıralı bir dile dönüştürmeyi keşfedeceksiniz. Dönem boyunca adım adım geliştireceğiniz **Kod-Asistan** projesiyle kullanıcının sorunlarını mantıksal adımlarla çözen işlevsel bir dijital yardımcı tasarlayacaksınız. Bilgisayarlara verilecek talimatları açık, sıralı ve eksiksiz biçimde yapılandırmaya hazır mısınız?

HAZIRLIK ÇALIŞMALARI

1. Bir arkadaşınıza en sevdiği yemeğin tarifini verirken adımların sırasını karıştırmanın veya bir adımı atlamanın (örneğin tencereye sadece makarnayı koyup su eklemeden ocağı yakmanın) ne gibi sonuçları olabilir? Düşüncelerinizi arkadaşlarınızla paylaşınız.
2. Verilen anahtar kavramın size neleri çağrıştırdığını düşününüz. Bu kavramla ilgili aklınıza gelen en fazla 5 kelimeyi ilgili kutucuklara yazınız. Ardından bu kelimeleri kullanarak kavram hakkındaki ön bilgilerinizi yansıtan anlamlı bir cümle kurunuz.

Anahtar Kavram: PROBLEM ÇÖZME

a)

b)

c)

ç)

d)

İlgili Cümlem:

.....

.....

.....

.....

.....

.....

1.1. PROBLEM ANALİZİ VE BİLEŞENLERİ

Programlama sürecinin ilk ve en önemli adımı, karşılaşılan problemin doğru bir şekilde analiz edilmesidir. Bilgisayar bilimlerinde bir sorunun çözümüne ulaşmak için öncelikle hangi verilere (girdi) ihtiyaç duyulduğunun belirlenmesi, bu veriler üzerinde yapılacak matematiksel veya mantıksal işlemlerin tanımlanması ve elde edilmesi beklenen sonucun (çıktı) netleştirilmesi gerekir. Bu sürece **problem analizi** adı verilir. Örneğin bir okul kantininde yaşanan sıra bekleme sorununu çözmek için öğrenci sayısı ve alışveriş süreleri girdi olarak belirlenebilir. Ardından uygun sıra ve ödeme düzeninin oluşturulmasına yönelik işlemler gerçekleştirilir. Bu işlemler sonucunda ortalama bekleme süresinin hesaplanması veya önerilen sıra düzeninin oluşturulması çıktı olarak değerlendirilir.

1.1.1. Algoritma ve Tasarım Kriterleri

Bilgisayar bilimlerinde, yapılan analizlerin eksiksiz bir şekilde hayata geçirilebilmesi ve çözüm yollarının kurgulanabilmesi için algoritma adı verilen tasarımdan yararlanılır. Algoritma, belirli bir problemi çözmek veya hedeflenen bir amaca ulaşmak için tasarlanan, açık ve sıralı adımlardan oluşan mantıksal bir süreçtir. Bilgisayarlar kendilerine tanımlanan komutlar ve kurallar doğrultusunda işlem yaptığından, algoritmanın her adımı anlaşılır, uygulanabilir ve yoruma kapalı olmalıdır.

Belirli bir problemi çözmek üzere tasarlanan bir sürecin algoritma olarak değerlendirilebilmesi için üç temel ilkeyi karşılaması beklenir. İlk olarak algoritmanın bir başlangıç noktası bulunmalı ve tanımlanan koşullar altında belirli bir süre sonra tamamlanmalıdır. Bu özellik **sonluluk** olarak adlandırılır. İkinci olarak her işlem adımı tek bir anlama gelecek biçimde açık ve kesin olarak ifade edilmelidir. Bu özellik **belirlilik** veya **kesinlik** olarak adlandırılır. Son olarak algoritmadaki adımlar, uygulanabilir ve izlenebilir işlemlerden oluşmalıdır. Bu özellik ise **etkinlik** olarak ifade edilir.

OKU-YORUM

Bilimsel Mirasın Temel Taşı: El-Harezmi

Bugünkü dünyada akıllı telefonlardan otonom araçlara, yapay zekâ sistemlerinden veri analiz araçlarına kadar kullanılan tüm modern teknolojilerin kalbinde tek bir temel kavram yatmaktadır: algoritma. Peki, dijital çağın mimarisini oluşturan bu kavramın kökenlerinin 9. yüzyıl Bağdat'ına, Doğu İslam dünyasının en büyük bilim merkezlerinden biri olan Beytülhikme'ye (Bilgelik Evi) dayandığını biliyor muydunuz?

Modern bilgisayar biliminin temellerini atan büyük İslam bilgini, matematikçisi ve astronomu **El-Harezmi (780-850)**, insanlık tarihinin en büyük mantıksal devrimlerinden birine imza atmıştır. Onun Latinceye **Algoritmi de numero Indorum** olarak tercüme edilen eseri hem bugün kullanılan **sıfır (sifr)** kavramını dünyaya tanıtmış hem de bilim dünyasına **adım adım ilerleme** yöntemini kazandırmıştır. Batı dillerindeki algorism ve zamanla dönüşen **algorithm (algoritma)** kelimesi de El-Harezmi'nin adının Avrupalı bilim insanları tarafından telaffuz edilemeyip Latinceye **algoritmi** şeklinde uyarlanmasıyla literatüre girmiştir.

Algoritmik düşüncenin tarihsel kökleri, El-Harezmi'nin kurguladığı bu adım adım ilerleme

OKU-YORUM

yöntemine dayanır. Bu yöntem; karmaşık ve çözülemez görünen büyük bir problemi mantıklı, sonlu ve birbirini takip eden küçük adımlara bölerek neticeye ulaştırmayı hedefler. Bilgisayarların henüz icat edilmediği bir çağda insan zihninin çalışma prensiplerini ve mantıksal çözüm yollarını formüle eden bu yaklaşım, bugünkü programlama dillerinin doğrudan temelini oluşturur.

El-Harezmi'nin cebirsel denklemleri çözmek için hiyerarşik bir düzene bağladığı bu kurallar bütünü, bir bilgisayarın verileri işleme ve problem çözme mantığıyla birebir aynıdır. Türk-İslam medeniyetinin ve millî teknoloji tarihinin köklerinde yer alan bu eşsiz miras, bugünün yazılım mimarilerine asırlar öncesinden rehberlik etmeye devam etmektedir (Görsel 1.1).



Görsel 1.1: Temsili El-Harezmi'nin algoritma mantığının modern yapay zekâ ve bilgisayar teknolojilerine dönüşümü

Düşünelim-Tartışalım

Okuduğunuz "Bilimsel Mirasın Temel Taşı: El-Harezmi" metninden hareketle aşağıdaki 3-2-1 Çıkış Kartı'nı doldurunuz.

3 BİLGİ: El-Harezmi'nin bilim dünyasına katkıları ve algoritma kavramının kökeni hakkında öğrendiğiniz en önemli üç bilgiyi yazınız.

.....

.....

.....

2 SORU: Al-Harezmî'nin geliştirdiği algoritma yönteminin günümüzdeki akıllı telefonlarda veya yapay zekâlarda nasıl kullanıldığına dair merak ettiğiniz iki soruyu yazınız.

.....

.....

1 YORUM: Türk-İslam medeniyetinin köklerinde yer alan algoritma mantığının, bugünün modern yazılım mimarilerine rehberlik etmesinin (millî teknoloji tarihi açısından) sizin için ne ifade ettiğini kendi cümlelerinizle yorumlayınız.

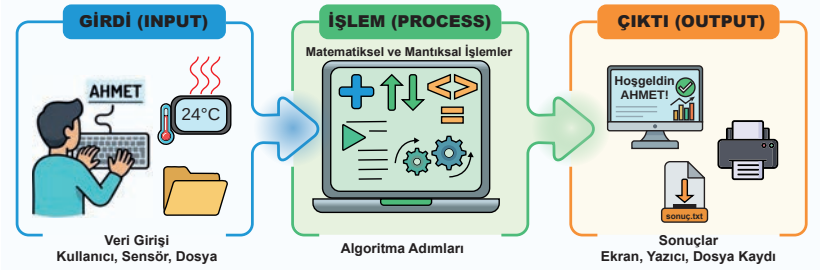
.....

1.1.2. Problemi Bileşenlerine Ayırma

Yazılım dünyasında karşılaşılan bazı problemler, tek adımda çözülemeyecek kadar büyük ve karmaşık olabilir. Bu tür durumlarda **Problemi Bileşenlerine Ayırma (Decomposition)** yaklaşımı kullanılır. Bu yaklaşım, büyük bir problemi daha küçük, anlamlı ve kolay yönetilebilir alt problemlere veya bileşenlere ayırma sürecidir. Her bileşenin ayrı ayrı analiz edilmesi, hata riskini azaltır; sistemin geliştirilmesini, test edilmesini ve gerektiğinde güncellenmesini kolaylaştırır. Örneğin kapsamlı bir dijital oyun tasarlanırken grafiklerin, kullanıcı kontrollerinin, seslerin, puanlama sisteminin ve oyun senaryosunun aynı anda ele alınması karmaşaya ve mantık hatalarına yol açabilir. Bunun yerine problem farklı bileşenlere ayrılabilir. Karakter hareketleri bir bileşen, ses ve müzik işlemleri başka bir bileşen, puanlama sistemi ise ayrı bir bileşen olarak tasarlanabilir. Her bileşen kendi görevi doğrultusunda geliştirilip test edildikten sonra diğer bileşenlerle bir araya getirilir. Böylece sistemin genel yapısını anlamak, hataları belirlemek ve yazılımı düzenli biçimde geliştirmek daha kolay hâle gelir.

1.1.3. Girdi-İşlem-Çıktı Modeli

Bilgisayar programları, karmaşık problemleri çözmek için verileri belirli bir düzende işler. Bu düzenin temel yapı taşı **Girdi-İşlem-Çıktı (GİÇ)** modelidir (Görsel 1.2). Algoritmik düşünme becerisinin temeli, bir problemi bu üç bileşene doğru şekilde ayrıştırabilmektir.



Görsel 1.2: Girdi-İşlem-Çıktı modeli

Sistemin çalışabilmesi için öncelikle dış dünyadan bazı başlangıç verilerinin alınması gerekir. Kullanıcının klavyeden girdiği bir isim, sensörden okunan sıcaklık değeri veya bir dosyanın içeriği gibi sisteme aktarılan her türlü ham veri **girdi (input)** olarak değerlendirilir. Sisteme alınan bu verilerin hedeflenen sonuca ulaşması için arka planda çeşitli matematiksel veya mantıksal denetimler gerçekleştirildiği aşama **işlem (process)** sürecini oluşturur. Bu işlemlerin başarıyla tamamlanmasının ardından elde edilen ve ekrandaki mesaj veya yazıcı dökümü gibi kullanıcıya sunulan nihai ürünler ise **çıktı (output)** olarak sisteme yansıtılır. Bu model, günlük hayattaki bankamatikten para çekme senaryosuyla somutlaştırılabilir. Kullanıcının cihaza taktığı kart bilgileri, tuşladığı şifre ve çekmek istediği para miktarı sisteme verilen birer girdidir. Sistem, şifrenin doğruluğunu teyit edip çekilecek tutarı mevcut bakiyeden düşerek işlem aşamasını tamamlar. Para haznesinden verilen banknotlar ve ekrandaki yeni bakiye bilgisi ise sürecin başarılı bir şekilde sonlandığını gösteren çıktılardır.

1. UYGULAMA

Uygulamanın Adı: PROBLEMİ PARÇALA

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: Problemi alt parçalara ayırmak ve GİÇ mantığıyla analiz etmek

Problem Senaryosu: Günlük hayatta sıkça karşılaşılan yiyecek ve içecek otomatları, arka planda karmaşık karar ve kontrol mekanizması işletir. Sisteme paranın atılması, sahtelik kontrolü, stok sorgulaması, bakiye kıyaslaması, para üstünün hesaplanması gibi birçok adım ardışık olarak gerçekleşir. Kod-Asistan projesi için bu yapıda bir otomat modülü geliştirilecektir ancak bu karmaşık işlem adımlarının tek parça ve plansız biçimde tasarlanması, mantık hatalarının gözden kaçmasına ve sistemin test edilmesinin zorlaşmasına neden olabilir. Meydana gelebilecek bu problemi çözmek ve sistemin kararlı çalışmasını sağlamak için kurgulanan karmaşık yapının işlem basamaklarına göre doğru alt parçalara ayrılması gerekmektedir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: GİÇ Analiz Tablosu

a) Verilen GİÇ tablosunu inceleyiniz ve bir **otomat makinesinin çalışma algoritmasını** düşünerek çözüm adımlarınızı ayrılmış boşluklara yazınız.

Sistem Bileşeni	Analiz Soruları	Çözüm Adımları
Girdi	Otomat makinesinin kullanıcıdan veya sensörlerden **aldığı veriler ve sinyaller** nelerdir?	1. ..... 2. ..... 3. .....
İşlem	Sistem girdileri aldıktan sonra hedefe ulaşmak için arka planda **hangi matematiksel ve mantıksal kontrolleri** sırayla yapar?	1. ..... 2. ..... 3. .....
Çıktı	Tüm kontrol ve işlem adımları sorunsuz tamamlandığında kullanıcıya veya sisteme **sunulan somut sonuçlar** nelerdir?	1. ..... 2. ..... 3. .....

1. UYGULAMA

2. Aşama: Eş-Geliştirici Yapay Zekâ ile Karşılaştırmalı Analiz

Ön Bilgi: Bilişim dünyasında yapay zekâ, tek başına kod yazan bir otomat değil fikirlerinizin test edildiği bir **eş-geliştirici** (**pair-programmer)** olarak konumlandırılmalıdır. Kendi kurguladığınız otomat algoritmasını test etmek için verilen adımları uygulayınız.

a) Bir yapay zekâ aracına şu komutu veriniz: "Senden bir otomat makinesinin çalışma algoritmasını oluşturmanı istiyorum. Adımları, Girdi-İşlem-Çıktı şeklinde belirle ve içine mantıksal bir karar yapısı ekle."

b) Yapay zekânın ürettiği adımları, kendi doldurduğunuz GİÇ matrisiyle yan yana getirerek aşağıdaki analiz sorularını yanıtlayınız.

- Yapay zekânın ürettiği çözümde eksik, genel bırakılmış veya sizin çözümünüzden farklı ifade edilmiş adımlar var mıdır?

  Cevap: .....

- Yapay zekânın belirlediği karar noktası, sizin belirlediğiniz koşula göre daha açık, uygulanabilir veya amaca uygun mudur? Neden?

  Cevap: .....

Değerlendirme: Yapacağınız çalışma, karekodda yer alan Dereceli Puanlama Anahtarı'na göre değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Kendi öğrenme sürecinizi değerlendirmek için diğer karekodda yer alan Öz Değerlendirme Formu'nu doldurunuz.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Öz Değerlendirme Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Bilgisayar bilimlerinde karmaşık sistemler tasarlanırken Girdi-İşlem-Çıktı modeli tek bir parça hâlinde uygulanmaz. Devasa ve çok değişkenli problemlerle karşılaşıldığında bu model, **Problemi Bileşenlerine Ayırma** yaklaşımıyla birleştirilerek kullanılır. Akıllı bina sistemleri, otonom araçlar veya gelişmiş otopark sistemleri gibi yoğun sensör verisi barındıran mimarilerin arka planındaki veri akışını zihinsel olarak küçük parçalara ayırmak, algoritmik düşünme sürecini büyük ölçüde kolaylaştırır. Yazılım geliştiriciler, büyük bir problemi daha küçük GİÇ bloklarına bölerek her bir alt görevi ayrı ayrı analiz eder ve böylece sistemin geliştirilmesini ve test edilmesini kolaylaştırarak hata riskinin azaltılmasına yardımcı olur.

1. SIRA SİZDE

Modern bir alışveriş merkezinin Akıllı Otopark Giriş-Çıkış Sistemi, arka planda otomatik karar mekanizmaları işletir. Araç yanaştığında girişte plaka kameradan okunur, içeride boş yer olup olmadığı (doluluk) kontrol edilir ve giriş zamanı sisteme kaydedilir. Araç ayrılmak istediğinde ise çıkışta kalış süresi hesaplanır, üyelik ve ücretsiz kullanım hakkı denetlenir, gerekiyorsa otopark ücreti belirlenir ve ödeme sonrasında bariyerin açılması tetiklenir. Bu karmaşık yapının, rehber bir şablon veya yapay zekâ desteği olmaksızın zihinsel şekilde alt parçalara ayrılması beklenmektedir.

Verilen problem senaryosunu Problemi Bileşenlerine Ayırma yaklaşımını kullanarak analiz ediniz ve aşağıdaki adımları uygulayınız.

a) Bu sistemin çalışabilmesi için dış dünyadan alınması gereken en az 3 farklı girdi verisini belirleyip kâğıdınıza yazınız.

b) Sistemin arka planda yapması gereken en az 2 matematiksel veya mantıksal işlem adımını sıralayınız.

c) Süreç başarıyla tamamlandığında dış dünyaya aktarılacak en az 2 çıktı sonucunu netleştiriniz.

Güvenlik Notu: Girdi verilerini belirlerken gerçek plaka veya kişisel veri kullanmayınız, temsilî değerler veya veri maskeleme yöntemlerini tercih ediniz.

1.2. AKIŞ ŞEMASI (FLOWCHART)

Bir problemi çözmek için zihinde kurgulanan algoritmaları doğrudan bilgisayar diline (kodlara) çevirmeye çalışmak, tıpkı bir mimarın elinde hiçbir çizim veya proje olmadan bina inşa etmeye başlamasına benzer. Tasarlanan algoritmaların bilgisayar diline dönüştürülmeden önce insanlar tarafından kolayca anlaşılabilmesi ve test edilebilmesi gerekir. Kodlamaya geçilmeden önce çözüm mantığının belirli standartlarla kâğıt üzerinde modellenmesi, karmaşık projelerdeki mantık hatalarının henüz yolun başındayken fark edilmesini sağlar.

1.2.1. Sözde Kod ve Elle Çalıştırma

Bilgisayar bilimlerinde algoritmaları modellemek için en sık başvurulan yöntemlerden biri **sözde kod**dur. **Sözde kod (pseudo code)**; herhangi bir programlama dilinin (Python, C++, Go vb.) katı yazım kurallarına bağlı kalmadan bir algoritmanın günlük konuşma diline yakın, net ve komut tabanlı ifadelerle kâğıda dökülmüş hâlidir. Sözde kodlar, programcının mantık hatalarına odaklanmasını kolaylaştırır.

Dry run; tasarlanan bir algoritmanın veya sözde kodun bilgisayar ortamına aktarılmadan önce sadece kâğıt ve kalem yardımıyla elle, adım adım işletilmesi sürecidir. Kodlamaya geçmeden önce yapılan elle çalıştırma, değişkenlerin aldığı değerleri ve algoritmanın izlediği yolları adım adım incelemeyi; hesaplama, karar ve döngü yapılarındaki mantık hatalarını erken aşamada fark etmeyi sağlar. Bir dry run çalışmasında öncelikle algoritmada değerleri sürekli değişebilen her bir veri (değişken) için bir takip

tablosu oluşturulur. Ardından algoritmanın veya sözde kodun ilk adımından başlanarak her bir işlem adımında değişkenlerin değerlerindeki değişimler satır satır bu tabloya kaydedilir. Son aşamada ise algoritmanın mantıksal akışı son adıma kadar izlenerek beklenen çıktıya ulaşılıp ulaşılmadığı kontrol edilir.

1. ÖRNEK

İKİ SAYININ TOPLAMINI BULMA

Bir kullanıcının sisteme ardı ardına girdiği iki sayısal değerin toplanması ve sonucun ekrana yazdırılması istenmektedir. Bu mantıksal akışı gerçekleştirmek için dışarıdan alınacak birinci sayı (S1) ve ikinci sayı (S2) algoritmaya **girdi** olarak sunulurken bu iki sayının hesaplanan toplamı (Toplam) ise sistemin nihai **çıktısı** olarak modellenmiştir.

Sözde Kod İfadesi

1. Adım: BAŞLA.

2. Adım: Birinci sayıyı (S1) kullanıcıdan al.

3. Adım: İkinci sayıyı (S2) kullanıcıdan al.

4. Adım: Toplam = S1 + S2 işlemini yap.

5. Adım: Toplam sonucunu ekrana yazdır.

6. Adım: DUR.

Elle Çalıştırma Tablosu

Örnek Girdiler: S1 = 15, S2 = 25

Beklenen Çıktı: Toplam = 40

Adım	İşlem	S1	S2	Toplam	Açıklama
1	BAŞLA.	-	-	-	Algoritma başlar.
2	S1 değerini oku.	**15**	-	-	Kullanıcı 15 sayısını girer.
3	S2 değerini oku.	15	**25**	-	Kullanıcı 25 sayısını girer.
4	Toplam = S1 + S2	15	25	**40**	15 + 25 = 40 işlemi gerçekleşir.
5	Toplam değerini yazdır.	15	25	40	Ekrana 40 sonucu yazılır.
6	DUR.	15	25	40	Algoritma başarıyla sonlanır.

Algoritmalar; her zaman yukarıdan aşağıya düz bir çizgide ilerlemez, günlük hayatta olduğu gibi belirli koşullara göre farklı yollara sapabilir. Yazılımcılar, programı bu yol ayrımlarında doğru yönlendirebilmek için iki veriyi birbiriyle kıyaslayan ve aralarındaki mantıksal ilişkiyi belirleyen **karşılaştırma operatörlerini** kullanırlar. Bir karşılaştırma ifadesinin doğrudan sonucu Doğru (True) veya Yanlış (False) mantıksal değeridir.

Algoritma tasarımlarında en sık başvurulan karşılaştırma operatörleri ve bunların çalışma mantıkları Tablo 1.1'de özetlenmiştir.

Tablo 1.1: Karşılaştırma Operatörleri ve Kullanım Örnekleri

Operatör	Anlamı	Örnek Kullanım	Bilgisayar Dilindeki Karşılığı ve Açıklama
==	Eşittir	tahmin == 1234	Girilen tahmin değeri, sistem şifresine (1234) eşitse sonuç **Doğru** olur.
!=	Eşit Değildir	sifre != "123"	Şifre, "123" değerine eşit değilse sonuç **Doğru** olur.
>	Büyüktür	yas > 18	Yaş değeri, 18'den büyükse sonuç **Doğru** olur.
<	Küçüktür	sicaklik < 0	Sıcaklık, 0'dan küçükse sonuç **Doğru** olur.
>=	Büyük Eşittir	puan >= 50	Puan, 50 veya 50'den daha büyükse sonuç **Doğru** olur.
<=	Küçük Eşittir	hiz <= 90	Hız, 90 veya 90'dan daha küçükse sonuç **Doğru** olur.

2. ÖRNEK

YAŞ KONTROLÜ VE EHLİYET ALABİLME

Sürücü belgesi başvuru sisteminde, adayın doğum yılı bilgisinden hareketle mevcut yaşı hesaplanmakta ve yasal sınır olan 18 yaşını doldurup doldurmadığı kontrol edilmektedir. Bu mantıksal akışı gerçekleştirmek için algoritmaya sunulan Sistem Tarihi (ST) ve adayın Doğum Yılı (DY) bilgileri temel girdileri oluşturur. Yaşın uygunluk durumuna göre arka planda üretilen "Ehliyet alabilirsiniz." veya "Ehliyet almak için henüz küçüksünüz." durum mesajları ise sistemin nihai çıktılarını ifade eder.

Not: Sistem tarihi (ST) algoritmalarda dinamik bir değişken olarak ele alınmalıdır. Bu örnekteki 2026 yılı, yalnızca aritmetik gösterim amacıyla verilmiştir. Bu hesaplama yaklaşık yaşı verir; gerçek yaşın yasal olarak belirlenmesi için doğum tarihi ile güncel tarih gün, ay ve yıl olarak birlikte değerlendirilmelidir.

Sözde Kod İfadesi

1. Adım: BAŞLA.

2. Adım: Sistem Tarihi (ST) değerini oku.

3. Adım: Kullanıcının Doğum Yılı (DY) değerini oku.

4. Adım: Yaş = ST - DY işlemini yap.

5. Adım: Eğer Yaş >= 18 ise
"Ehliyet alabilirsiniz." mesajını yazdır.
DEĞİLSE
"Ehliyet almak için henüz küçüksünüz." mesajını yazdır.

6. Adım: DUR.

Elle Çalıştırma Tablosu

Örnek Girdiler: Sistem Tarihi (ST) = 2026, Doğum Yılı (DY) = 2005

Beklenen Çıktı: Ehliyet alabilirsiniz.

2. ÖRNEK

Adım	İşlem	ST	DY	Yaş	Koşul Denetimi (Yaş >= 18)	Çıktı	Açıklama
1	BAŞLA.	-	-	-	-	-	Algoritma başlar.
2	ST oku.	**2026**	-	-	-	-	Sistem tarihi alınır.
3	DY oku.	2026	**2005**	-	-	-	Kullanıcı 2005 sayısını girer.
4	Yaş = ST - DY	2026	2005	**21**	-	-	2026 - 2005 = 21
5	Yaş >= 18?	2026	2005	21	**Evet**	**Ehliyet alabilirsiniz.**	Koşul doğru → mesaj yazdırılır.
6	DUR.	2026	2005	21	-	-	Algoritma başarıyla sonlanır.

Örnek Girdiler: Sistem Tarihi (ST) = 2026, Doğum Yılı (DY) = 2010

Beklenen Çıktı: Ehliyet almak için henüz küçüksünüz.

Adım	İşlem	ST	DY	Yaş	Koşul Denetimi (Yaş >= 18)	Çıktı	Açıklama
1	BAŞLA.	-	-	-	-	-	Algoritma başlar.
2	ST oku.	**2026**	-	-	-	-	Sistem tarihi alınır.
3	DY oku.	2026	**2010**	-	-	-	Kullanıcı 2010 sayısını girer.
4	Yaş = ST - DY	2026	2010	**16**	-	-	2026 - 2010 = 16
5	Yaş >= 18	2026	2010	16	**Hayır**	**Ehliyet almak için henüz küçüksünüz.**	Koşul yanlış → mesaj yazdırılır.
6	DUR.	2026	2010	16	-	-	Algoritma başarıyla sonlanır.

Yaş kontrolü örneğinde algoritmanın sadece yazılımsal bir mesaj ürettiği görülür. Bununla birlikte karar yapıları sadece ekrana yazı yazdırmakla kalmaz; sensörlerden alınan verilere göre motor, fan, ısıtıcı, vana veya klima gibi aktüatörlerin çalışmasını kontrol edebilir. Karşılaştırma operatörlerinin akıllı ev sistemlerinde nasıl çalıştığı 3. Örnek üzerinden şöyle incelenir:

3. ÖRNEK

SICAKLIK KONTROLÜ ALGORİTMASININ SÖZDE KODU

Bir akıllı sistemin ortam sıcaklığı 20 °C'un altına düştüğünde ısıtıcıyı otomatik olarak devreye alması, 20 °C ve üzerinde ise sistemi beklemede tutarak kullanıcıyı bilgilendirmesi istenmektedir. Bu akışı yöneten sözde kod yapısı şöyledir:

Adım	Sözde Kod İfadesi
1.	BAŞLA.
2.	Ortam sıcaklık değerini oku (sicaklik).
3.	EĞER sicaklik < 20 İSE: Isıtıcıyı çalıştır. DEĞİLSE: Ekrana "Sıcaklık 20 °C'nin altında değildir." yazdır.
4	DUR.

Şu ana kadar incelenen ilk üç algoritmada sürecin her koşulda DUR adımına ulaşarak doğrusal şekilde sonlandığı görülmüştür. Ancak 4. Örnek'teki temel şifre kontrolünde olduğu gibi algoritmalar her zaman tek seferde bitmez. Kullanıcı şifreyi yanlış girmeye devam ettiği sürece sistem DUR adımına ulaşamaz ve sürekli başa döner. Bir algoritmanın kendi içinde geriye dönerek işlemi tekrarlamasına **döngüsel akış** denir.

4. ÖRNEK

HATALI GİRİŞ KONTROLÜ ALGORİTMASI

Bir kullanıcı panelinde sisteme erişmek isteyen kişinin girdiği şifre bilgisi kontrol edilmektedir. Sistem, kullanıcı doğru şifreyi yazana kadar geçişe izin vermemektedir ve her hatalı denemede bir uyarı mesajı üreterek kullanıcıyı tekrar şifre giriş adımına yönlendiren döngüsel akış işletmektedir.

Adım	Açıklama
1.	BAŞLA.
2.	Sistem şifresini (sifre = "1234") belirle.
3.	Şifre al (tahmin).
4	EĞER tahmin == sifre İSE: Ekrana "Giriş Başarılı, Hoş Geldiniz!" yazdır. ADIM 7'ye GİT.
5	DEĞİLSE: Ekrana "Hatalı şifre girdiniz, tekrar deneyiniz." yazdır.
6	ADIM 3'e GİT.
7	DUR.

2. UYGULAMA

HATA AVCILIĞI (KODU GÖZDEN GEÇİRME)

Uygulamanın Süresi: 20 dakika

Uygulamanın Amacı: Algoritma kabul edilebilme kriterlerini temel alarak tasarlanmış bir sözde kodun yapısal ve mantıksal kusurlarını denetleme, satır satır kod gözden geçirme (code review) ve güvenli çözüm üretmek

Problem Senaryosu: Kod-Asistan projenizin **Kullanıcı Giriş Modülü** için başka bir yazılım ekibi tarafından aceleyle bir sözde kod tasarlanmıştır. Bu algoritmanın elle çalıştırma (dry run) sırasında mantıksal hata içerdiği ve programın DUR adımına hiçbir koşulda ulaşamayarak sonsuz döngüye girdiği fark edilmiştir. Projenin kararlılığı için bu hataların saptanması ve algoritmanın yapılandırılmış kurallara göre yeniden düzenlenmesi gerekmektedir. Kullanıcı Giriş Modülü için tasarlanan ve denetlenmesi gereken hatalı algoritma adımları aşağıda sunulmuştur:

İncelenecek Hatalı Algoritma

1. Adım: BAŞLA.

2. Adım: Sistem şifresini (sistem_sifresi = "1234") belirle.

3. Adım: Kullanıcıdan şifre bilgisini al (sifre).

4. Adım: EĞER sifre == sistem_sifresi İSE ekrana “Giriş başarılı.” yazdır.

5. Adım: Ekrana “Hatalı şifre girdiniz, tekrar deneyiniz.” yazdır ve 3. Adım’a git.

6. Adım: DUR.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Algoritmayı Ölçütlere Göre Denetleme

a) İncelenecek hatalı algoritmayı, algoritma kabul edilebilme kriterlerine göre satır satır denetleyiniz.

b) Tespit ettiğiniz yapısal ve mantıksal kusurları verilen analiz alanına gerekçeleriyle yazınız.

Hata Analiz Raporunuz

Algoritmadaki En Temel Akış Hatası: .....

Sonluluk Açısından İnceleme Sonucunuz:.....

Mantıksal Akış Açısından İnceleme Sonucunuz: .....

2. Aşama: Karar Verme ve Yargıda Bulunma

a) Bir yazılımcının en büyük bilişsel güçlerinden biri, karşılaştığı karmaşık mantık hataları karşısında pes etmeyip çözüme ulaşana kadar kararlılık göstermesidir. Yaptığınız denetim sürecinde gösterdiğiniz bu algoritmik kararlılıktan hareketle tasarlanan hatalı algoritmanın neden hiçbir zaman DUR adımına ulaşamayacağını 4. ve 5. adımlar arasındaki mantıksal çelişkiyi temel alarak dürüst ve kesin bir dille açıklayınız.

Mantıksal Kararlılık Raporunuz: .....

2. UYGULAMA

3. Aşama: Düzeltme Önerisi (Çözüm Üretme)

a) Algoritmanın kusursuz çalışması için 4. ve 5. adımları yeniden düzenleyiniz ve doğru sözde kodu aşağıya yazınız.

1. Adım: BAŞLA.
2. Adım: Sistem şifresini (sistem_sifresi = 1234) belirle.
3. Adım: Kullanıcıdan şifre bilgisini al (sifre).
4. Adım: .....
5. Adım: .....
6. Adım: DUR.

Değerlendirme: Yapacağınız çalışma, karekodda yer alan Dereceli Puanlama Anahtarı'na göre değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

1.2.2. Akış Şeması Sembolleri ve Görsel Modelleme

Tasarlanan algoritma adımlarının, tüm dünyada kabul görmüş standart geometrik semboller ve oklar kullanılarak görsel bir şema hâline getirilmesine **akış şeması (flowchart)** denir. Akış şemaları, bir programın mantıksal akışını bir bakışta anlamayı sağlar ve kodlama aşamasına geçmeden önce olası mantık hatalarının (sonsuz döngüler, eksik koşullar vb.) tespit edilmesini kolaylaştırır.

Algoritmaları görsel bir haritaya dönüştürürken kullanılan her sembol, günlük hayatta karşılaşılan bir eylemi veya karar anını temsil eder. Algoritmadaki her bir işlem türü için farklı bir sembol kullanılır ve bu sembolleri sadece şekil olarak değil işlevleri üzerinden tanımak, mantık kurmayı büyük ölçüde kolaylaştırır. Sürecin sınırlarını çizen oval (terminal) sembolü; sabah uyanmak, gece uyumak gibi bir işin başlangıç ve bitiş anlarını ifade ederek algoritmanın çerçevesini (BAŞLA ve DUR adımlarını) belirler.

Programın dış dünyayla iletişim kurduğu anlarda ise paralelkenar (giriş / çıkış) devreye girer. Bu sembol, birine adını sormak gibi bir veri alışverişini veya bir soruya cevap vermek gibi ekrana sonuç yazdırma işlemini simgeler. Herhangi bir matematiksel hesaplama yapılması veya bir eylemin gerçekleştirilmesi gerektiğinde çanta hazırlamak, matematik problemi çözmek gibi somut bir hareketi temsil eden dikdörtgen (işlem) sembolü kullanılır.

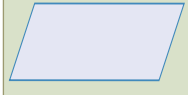
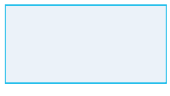
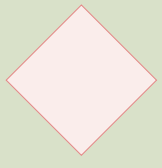
Gerçek dünya problemlerinin çözümü, sürecin belirli aşamalarında mantıksal bir karar verilmesini ve programın farklı alternatif yollara sapmasını gerektirir. Algoritmanın en kritik noktası olan ve günlük hayattaki yol ayrımlarını temsil eden eşkenar dörtgen (karar) sembolü bu amaçla kullanılır. Sistem bu noktada iki veriyi birbiriyle kıyaslayarak büyüklük veya küçüklük denetimi yapar ve sonucunda

her zaman Doğru (True) veya Yanlış (False) mantıksal değerini üretir. Örneğin "Hava yağmurlu mu?" sorusuna verilen cevaba göre şemsiye alıp almama kararı gibi programın akışı, üretilen bu sonuca göre oklar yardımıyla Evet veya Hayır yönünde ilgili işlem adımlarına yönlendirilir. Doğru bağlanan yön okları akışın izlenmesini kolaylaştırır; algoritmanın sonlanma koşulu doğru tasarlanmışsa süreç bitiş sembolüne ulaşır.

Tasarlanan algoritmalarda eşkenar dörtgen karar sembolünün içinin doldurulmasında önemli olan koşulun açık ve iki sonuçlu (Doğru/Yanlış) olmasıdır. Bir koşul ifadesi şemaya aktarılırken günlük konuşma dilindeki “Eğer” kelimesinin kullanılması teknik olarak yasak olmasa da genellikle şemayı sadeleştirmek için daha net bir dil tercih edilir. Örneğin sıcaklık değerini kontrol eden bir karar kutusuna uzun uzun “Eğer sıcaklık 20’den küçükse” yazmak yerine, akışı daha rahat okuyabilmek için çoğunlukla doğrudan “sicaklik < 20” şeklinde mantıksal bir ifade veya “sicaklik < 20 mi?” biçiminde kısa bir soru cümlesi yazılır.

Sembollerin doğru kullanımı, projenin diğer yazılımcılar tarafından da kusursuz anlaşılması için kritiktir. Programlama süreçlerinde sıklıkla karşılaşılan temel sembollerin geometrik isimleri, işlevleri ve bu işlevlerin mantığını kavramayı kolaylaştıracak günlük hayat benzetmeleri Tablo 1.2’de detaylıca gruplandırılmıştır. Bu algoritmik kararlılığın ve görsel modelleme standartlarının akıllı bir iklimlendirme sistemi üzerinde nasıl çalıştığı ise sıradaki örnek üzerinden incelenebilir.

Tablo 1.2: Temel Akış Şeması Sembolleri, İşlevleri ve Günlük Hayat Karşılıkları

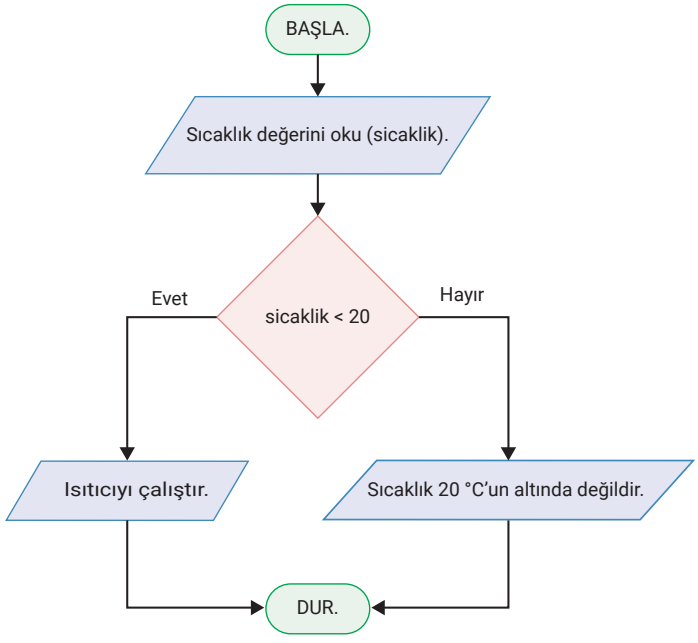
Sembol	Adı	İşlevi	Günlük Hayat Karşılığı
	Oval (Terminal)	Algoritmanın başlangıcını ve bitişini temsil eder. İçine BAŞLA veya DUR yazılır.	Bir işe başlama veya sonlandırma anıdır.
	Paralelkenar (Giriş / Çıkış)	Kullanıcıdan veri alınacağını veya ekrana bilgi yazdırılacağını belirtir.	Soru sormak veya cevap vermek içindir.
	Dikdörtgen (İşlem)	Matematiksel hesaplamalar ve değişken atamaları için kullanılır.	Bir eylemi gerçekleştirmek içindir.
	Eşkenar Dörtgen (Karar)	Bir koşulun kontrol edildiği noktadır. Sonuca göre akış Evet veya Hayır yönünde ilerler.	Yol ayrımında karar vermek içindir.
	Oklar (Akış Yönü)	Adımlar arasındaki ilerleme yönünü gösterir.	İzlenecek yolu gösteren oklardır.

5. ÖRNEK

SICAKLIK KONTROLÜ ALGORİTMASININ AKIŞ ŞEMASI

Akıllı bir iklimlendirme sisteminin karar mekanizması tasarlanacaktır. Ortam sıcaklığı verisi 20 °C'un altına düştüğünde ekrana "Isıtıcıyı çalıştır." uyarısının yazılması, değil ise ekrana "Sıcaklık 20 °C'nin altında değildir." uyarısı yazılması uygun akış şeması modeli Şema 1.1'de sunulmuştur.

Adım	İşlem
1	BAŞLA.
2	Ortam sıcaklık değerini al (sicaklik).
3	EĞER sicaklik < 20 İSE: Ekrana "Isıtıcıyı çalıştır." uyarısı yazdır.
4	DEĞİLSE: Ekrana "Sıcaklık 20 °C'nin altında değildir." uyarısı yazdır.
5	DUR.



Şema 1.1: Sıcaklık kontrolü algoritmasının akış şeması

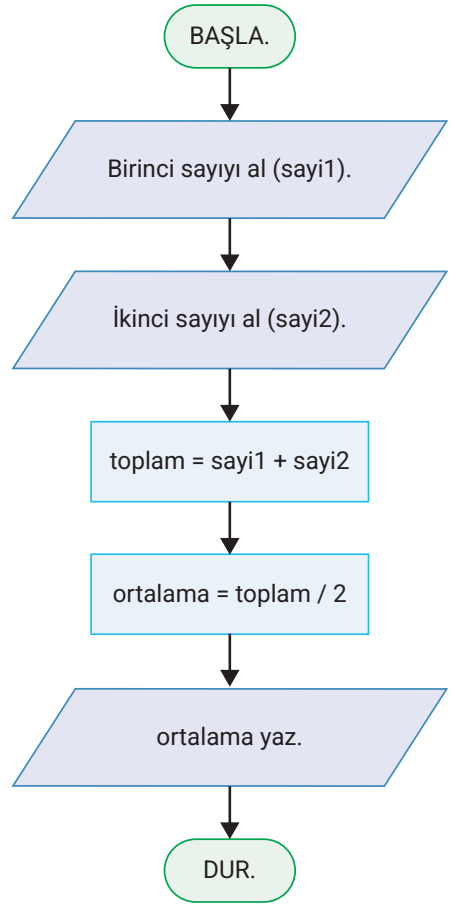
Karar yapılarının akış şemalarında elmas (eşkenar dörtgen) sembolüyle nasıl dallandığı ve sistemi nasıl yönlendirdiği görülür. Bununla birlikte bilgisayar programları sadece karar vermekle kalmaz, veriler üzerinde matematiksel işlemler de yürütür. Doğrusal veri işleme ve aritmetik hesaplama sürecinin işlem (dikdörtgen) sembolleriyle nasıl modellendiği 6. Örnek üzerinden şöyle incelenir:

6. ÖRNEK

İKİ SAYININ ORTALAMASINI BULMA ALGORİTMASININ AKIŞ ŞEMASI

Bir veri analiz modülünde sisteme aktarılan iki farklı sayısal değerin aritmetik ortalaması alınarak tek bir ortalama değeri elde edilecektir. Bu matematiksel sürecin ardışık adımlarını ve veri akış yönünü görselleştiren standart akış şeması modeli Şema 1.2'de yer almaktadır.

Adım	İşlem
1	BAŞLA.
2	Birinci sayıyı al (sayi1).
3	İkinci sayıyı al (sayi2).
4	toplam = sayi1 + sayi2
5	ortalama = toplam / 2
6	ortalama değerini ekrana yazdır.
7	DUR.



Şema 1.2: İki sayının ortalamasını bulma algoritmasının akış şeması

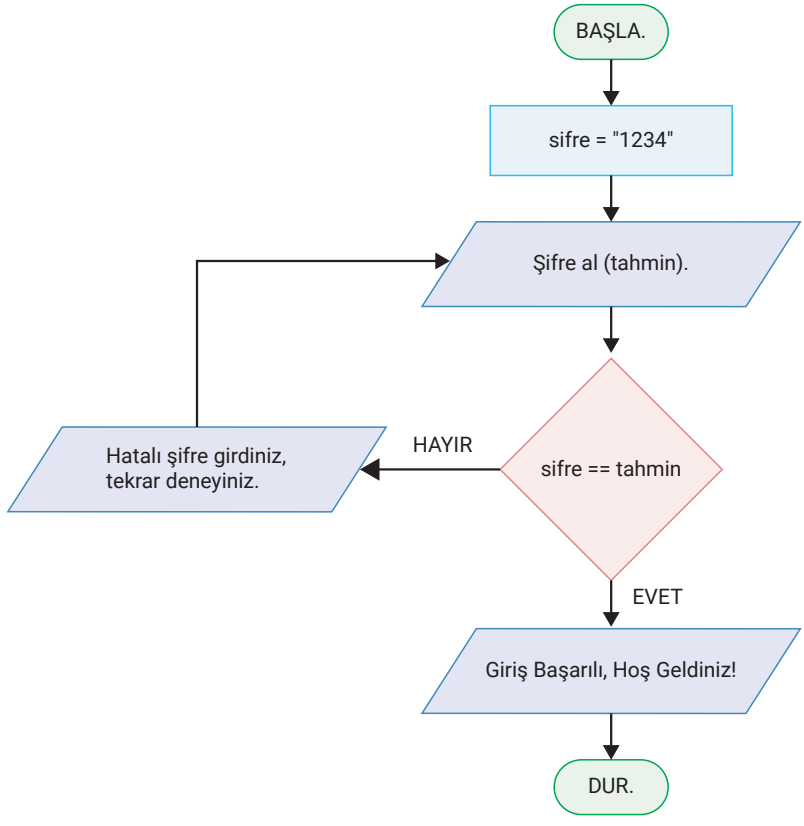
İncelenen akış şemalarında algoritmalar yukarıdan aşağıya doğru tek yönlü olarak ilerledi ve işlemi bitirdi. Gerçek dünya senaryolarında ise işlemler her zaman tek seferde sonlanmaz. Bazen kullanıcının hatalı bir veri girmesi durumunda akışın sonlanmak yerine geriye (yukarıya) dönerek işlemi tekrar etmesi gerekir. Akış yönü oklarının geriye dönerek nasıl güvenli bir kontrol mekanizması oluşturduğu 7. Örnek'teki şifre denetimi üzerinden şöyle incelenir:

7. ÖRNEK

HATALI GİRİŞ KONTROLÜ ALGORİTMASININ AKIŞ ŞEMASI

Kullanıcı doğrulama ekranında kurgulanan temel döngüsel giriş algoritmasının görsel haritası çıkarılacaktır. Doğru şifre girilene kadar süreci başa döndüren karar yapısını ve veri kontrollerini standart sembollerle gösteren akış şeması tasarımı Şema 1.3'te modellenmiştir.

Adım	İşlem
1	BAŞLA.
2	Sistem şifresini (sifre="1234")belirle.
3	Şifre al (tahmin).
4	EĞER tahmin == sifre İSE: Ekrana "Giriş Başarılı, Hoş Geldiniz!" yazdır. ADIM 7'ye GİT.
5	DEĞİLSE: Ekrana "Hatalı şifre girdiniz, tekrar deneyiniz." yazdır.
6	ADIM 3'e GİT.
7	DUR.



Şema 1.3: Hatalı giriş kontrolü algoritmasının akış şeması

2. SIRA SİZDE

Sayfa 22'de yer alan 1. Uygulama: Problemi Parçala uygulamasında bir otomat makinesinin çalışma mantığını GİÇ tablosuyla analiz etmiştiniz. Şimdi o analizde belirlediğiniz işlem adımlarını ve karar (koşul) yapılarını, öğrendiğiniz standart akış şeması sembollerini kullanarak kâğıt üzerinde çiziniz ve otomatın çalışma algoritmasını görselleştiriniz.

Yazılım tasarımları ilk seferinde kusursuz çalışmayabilir; karar sembolleri yanlış seçilebilir, yön okları hatalı bağlanabilir veya sıfır (0) gibi sınır durumlar gözden kaçabilir. Profesyonel bir yazılımcının temel yetkinliklerinden biri, tasarlanmış şemalardaki bu mantıksal kusurları saptayarak sistemi onarmaktır. Akış şemalarındaki görsel ve mantıksal hataların nasıl ayıklanacağı 3. Uygulama üzerinden adım adım çözümlenecektir.

3. UYGULAMA

Uygulamanın Adı : GÖRSEL HATA AVCILIĞI (DEBUGGING)

Uygulamanın Süresi: 20 dakika

Uygulamanın Amacı: Tasarlanan bir akış şemasındaki akış yönü oklarını, sembol hatalarını ve sınır durumlarını (sıfır olma durumu gibi) görsel şekilde analiz ederek mantıksal hataları saptamak ve uluslararası standartlara uygun şekilde yeniden kurgulamak

Problem Senaryosu: Bir yazılımcı, kullanıcının girdiği sayı 0'dan büyükse ekrana **Pozitif**, küçükse **Negatif** yazan bir programın akış şemasını tasarlamıştır. Test sürecinde yön oklarının yanlış bağlandığı, karar yapısının Evet / Hayır kollarının karıştığı ve kullanıcı **0 (Sıfır)** değerini girdiğinde sistemin bu değeri yanlışlıkla pozitif olarak sınıflandırdığı fark edilmiştir. Modülün kararlı çalışabilmesi için bu görsel ve mantıksal hataların saptanarak akış şemasının standartlara uygun şekilde onarılması gerekmektedir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

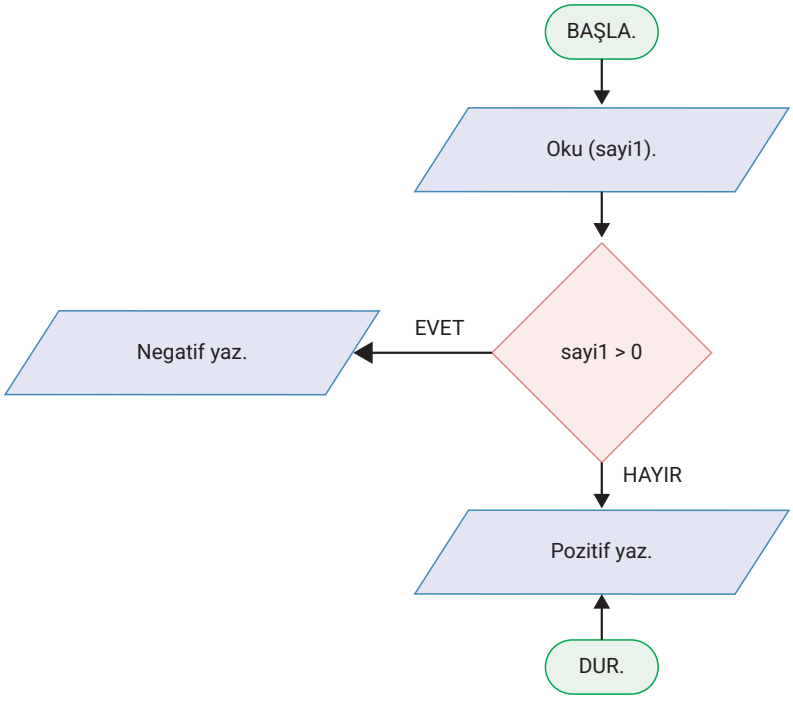
Uygulamanın Aşamaları

a) Verilen hatalı akış şeması taslağını standartlar ve mantıksal akış yönleri açısından detaylıca inceleyiniz.

b) Belirlediğiniz sembol, yön ve sınır durum hatalarını **Hata Takip Tablosu**'na hatanın nedeni ve düzeltme önerisiyle birlikte kaydediniz.

3. UYGULAMA

c) Hataları gidererek algoritmanın tüm durumları (pozitif, negatif, sıfır) kapsayan doğru ve güvenli akış şemasını standartlara uygun şekilde Şema 1.4'e göre yeniden çiziniz.



Şema 1.4: Hatalı akış şeması taslağı

Hata Takip Tablosu

Tespit Edilen Hata	Neden Hatalı?	Düzeltme Önerisi

Değerlendirme: Çalışmanız karekodda yer alan Gözlem Formu'yla değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Gözlem Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Yazılım projelerinde bazı işlem adımlarının belirli bir koşul gerçekleşene kadar ardı ardına tekrar etmesi gerekir. Her adımda kodları tek tek elle yeniden yazmak yerine bilgisayar bilimlerinde **döngü (loop)** adı verilen kontrol yapıları kullanılır. Sayaç kontrollü bir döngünün sağlıklı çalışabilmesi ve sonsuz bir döngüsel tıkanmaya girmemesi için şu üç temel bileşenin bulunması zorunludur:

Başlangıç Değeri: Döngünün hangi sayısal veya mantıksal değerden başlayacağını belirtir.

Döngü Koşulu: İşlemin devam edip etmeyeceğini denetleyen karar mekanizmasıdır. Koşul **Evet** ürettiği sürece döngü çalışır, **Hayır** ürettiğinde döngü güvenli bir şekilde sonlanır.

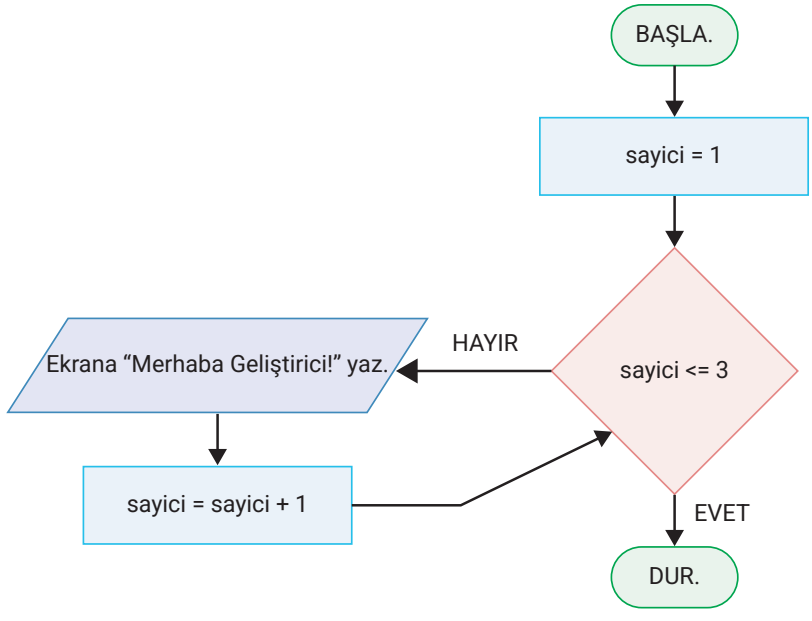
Sayıcı (Değişim Miktarı): Her turun sonunda döngünün kontrol değişkenini artıran veya azaltan matematiksel işlemdir.

Üç temel döngü bileşeninin bir akış şeması üzerinde nasıl bir araya geldiği ve sonsuz döngüye girmeden güvenli bir şekilde nasıl çalıştığı 8. Örnek üzerinden şöyle incelenir:

8. ÖRNEK

DÖNGÜSEL EKRAN YAZIMI

Kod-Asistan yazılımının kullanıcıyı karşılama modülünde ekrana ardı ardına üç kez "Merhaba Geliştirici!" mesajının yazdırılması istenmektedir. Bu işlemi doğrusal adımlarla tek tek yazmak yerine bir sayıcı değişkeni yardımıyla döngüsel olarak gerçekleştiren akış şeması tasarımı Şema 1.5'te modellenmiştir.



Şema 1.5: Karar yapılı döngüsel karşılama algoritması akış şeması

4. UYGULAMA

Uygulamanın Adı: ARDIŞIK SAYILARIN TOPLAMI

Uygulamanın Süresi: 20 dakika

Uygulamanın Amacı: Tekrarlanan süreçleri döngüsel mantıkla yapılandırıp akış şemasıyla modellemek ve değişkenlerdeki anlık değişimleri elle çalıştırma tekniğiyle adım adım takip etmek

Problem Senaryosu: Kod-Asistan yazılımının optimizasyon modülünde, dışarıdan belirlenen bir sınır değerine (n) kadar olan ardışık tam sayıların toplamına ihtiyaç duyulmaktadır. Bu işlem için arka planda bir sayıcının tutulup her adımda artırılması, değerlerin toplam değişkeninde biriktirilmesi ve koşul sağlandığında döngünün güvenli bir şekilde sonlandırılması kurgulanmıştır.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulamanın Aşamaları

a) Belirtilen senaryoya uygun şekilde döngü mantığı içeren algoritmanın eksik bırakılan 5 ila 9. adımları arasındaki sözde kod ifadelerini mantıksal bir hiyerarşiyle yazınız.

b) Tamamladığınız algoritmanın mantıksal akışını, standart geometrik semboller kullanarak tabloda ayrılan Akış Şeması alanına (Şema1.6) çiziniz.

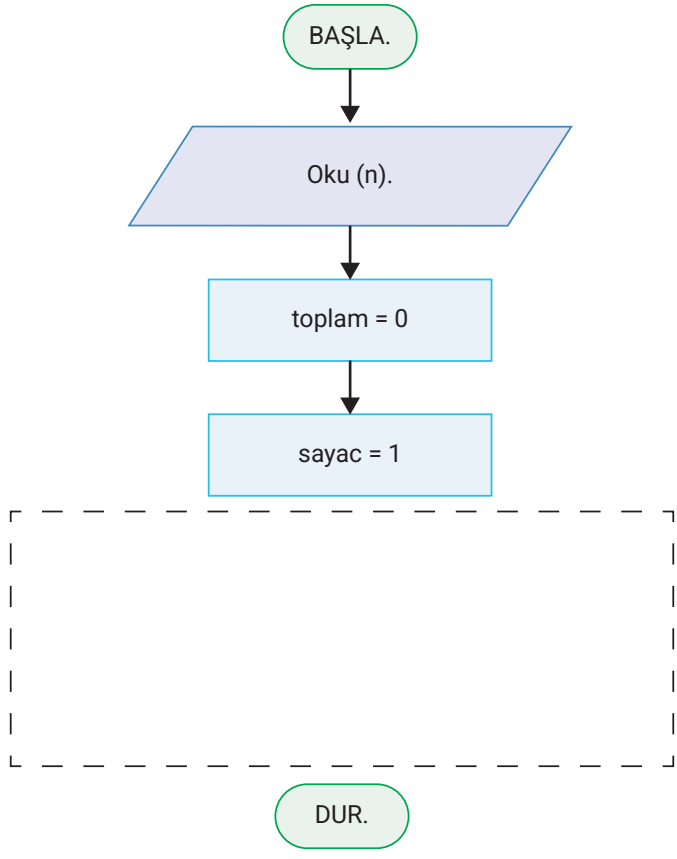
c) Tasarladığınız döngünün kararlılığını test etmek amacıyla n = 5 sınır değeri örneğini baz alınız.

ç) Bilgisayar kullanmadan sadece kâğıt-kalem yardımıyla **Elle Çalıştırma Tablosu** üzerinde sayıcı ve toplam değişkenlerinin her turda aldığı güncel değerleri satır satır takip ederek kaydediniz.

Algoritma

Adım	İşlem
1	BAŞLA.
2	n değerini oku.
3	toplam = 0
4	sayac = 1
5	
6	
7	
8	
9	
10	BİTİR.

4. UYGULAMA



Şema 1.6: Ardışık sayıların toplamı algoritmasının akış şeması

Elle Çalıştırma (n = 5 örneği)

Adım / Döngü	sayac Değeri	toplam Değeri	Koşul Denetimi (sayac > n)
Başlangıç	**1**	**0**	**-**
1. Adım	**1**	**1**	**Hayır (1 > 5 değil)**
2. Adım			
3. Adım			
4. Adım			
5. Adım			
Bitiş			

Değerlendirme: Çalışmanız karekodda yer alan Dereceli Puanlama Anahtarı'yla değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Gerçek dünya senaryolarında ise algoritmalar her zaman sabit sayılara göre sonlanmaz. Otonom sistemler ve akıllı altyapılar, sensör verilerini anlık takip ederek kritik bir eşik aşıldığında döngüyü anında kırmak (sonlandırmak) zorundadır. Döngülerin dış dünyadan gelen anlık verilere göre nasıl yönetilip acil durumlarda nasıl durdurulacağı 5. Uygulama üzerinden şöyle deneyimlenecektir:

5. UYGULAMA

Uygulamanın Adı: KRİTİK EŞİK

Uygulamanın Süresi: 20 dakika

Uygulamanın Amacı: Sensör verileriyle çalışan sınır denetimli bir döngü mekanizmasını sözde kodla yapılandırmak ve değişkenlerdeki anlık değişimleri elle çalıştırma tekniğiyle adım adım takip etmek

Problem Tanımı: Akıllı baraj otomasyonunda su seviyesi saatlik sensörlerle izlenmektedir. Su seviyesi 50 metreyi aşmadığı sürece kapaklar kapalı tutularak saat sayacı artırılır. Eşik aşıldığında ise döngü kırılarak tahliye saati ve kritik su seviyesi ekrana yazdırılmalıdır. Bu akışın sözde kodla modellenip kâğıt üzerinde test edilmesi kurgulanmıştır.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulamanın Aşamaları

a) Senaryoyu analiz ederek döngünün başlangıç değerini (saat = 1) ve barajın ilk su durumunu zihninizde yapılandırınız.

b) Katı kodlama kurallarına bağlı kalmadan günlük konuşma diline yakın ve sıralı emirlerden oluşan sözde kod adımlarını sıfırdan kâğıdınıza yazınız. Su seviyesi 50 metreyi aşınca döngüyü kırıp bitirecek mantıksal koşulu doğru basamağa yerleştirdiğinizden emin olunuz.

c) Yazdığınız sözde kodun kararlılığını test etmek amacıyla bilgisayarı açmadan sadece kâğıt ve kalem yardımıyla bir Elle Çalıştırma Takip Tablosu çiziniz.

ç) Tablonuzu test etmek için şu sıralı saatlik sensör verilerini simüle ediniz: 1. saat: 42 m, 2. saat: 48 m ve 3. saat: 55 m.

d) Değişkenlerin (saat, su_seviyesi) her adımda aldığı anlık değerleri satır satır tablonuza işleyiniz ve beklenen kritik çıkış saatine (3. saat) mantık hatası yapmadan ulaşıp ulaşmadığınızı denetleyiniz.

Değerlendirme: Çalışmanız karekodda yer alan Değerlendirme Formu'yla değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme Formu'na ulaşmak için karekodu okutunuz.

Öz Değerlendirme Formu'na ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

OKU-YORUM

Küçük Gauss'un Pratik Çözümü

Matematik tarihinin en büyük dehalarından biri kabul edilen Carl Friedrich Gauss, henüz ilkokul yıllarındayken öğretmeni sınıfı oyalamak için öğrencilerden 1'den 100'e kadar olan sayıları toplamalarını ister. Diğer öğrenciler sayıları tek tek toplamakla uğraşırken küçük Gauss sadece birkaç saniye içinde 5050 sonucunu bulur.

Gauss, sayı dizisinin başındaki ve sonundaki sayıları topladığında (1+100, 2+99, 3+98...) her zaman 101 sonucuna ulaşıldığını ve bu şekilde 50 tane çift olduğunu (50 x 101 = 5050) fark eder. Bu zekice yaklaşım, bugün bilgisayar biliminde karmaşık döngüler yerine neden matematiksel modellerin (algoritmaların) tercih edildiğine dair en güzel örneklerden biridir.

Düşünelim-Tartışalım

Bilgisayar bilimleri, mevcut çözümlerle yetinmeyip her zaman "Daha verimli bir yol var mı?" sorusunun peşinden giden güçlü bir entelektüel merak üzerine inşa edilir. Küçük Gauss'un ardışık sayıları tek tek toplamak yerine örüntüyü fark etmesini sağlayan derin merakından hareketle siz de bir algoritmayı tasarlarken binlerce kez döngü çalıştırmak ile tek bir matematiksel formül işletmek arasındaki farkı sorgulayınız.

Bu iki yaklaşımın yazılımın verimliliğine ve kararlılığına olan etkisini değerlendirerek aşağıdaki T-Tablosu'nun ilgili alanlarına gerekçeleriyle yazınız.

Binlerce Kez Döngü Çalıştırmak	Matematiksel Formül (Örüntü) İşletmek
Zaman Verimliliği	
İşlemci Yükü	
Yazılımın Kararlılığı	

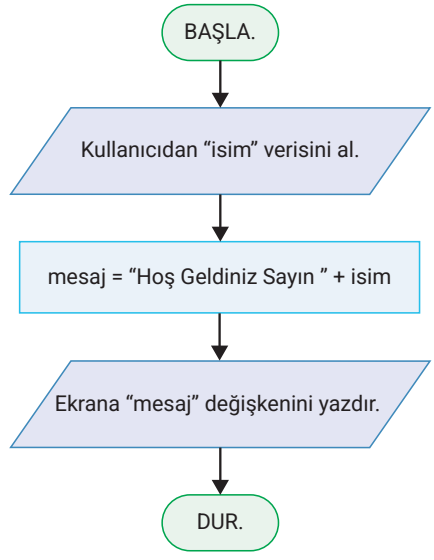
1.2.3. Algoritmalarda Metinsel Veriler ve İşlemler

Bilgisayarlar, sayılar dışında isim veya şifre gibi metinsel verileri de işler. Çift tırnak ("") içinde yazılan bu ifadelere **string (karakter dizisi)** denir. Metinsel veriler doğrudan aritmetik işlemlerde kullanılamaz. Sayıyı temsil eden bir metin, hesaplamada kullanılmadan önce uygun sayısal veri türüne dönüştürülmelidir. Yazılım dünyasında birden fazla metni yan yana getirmek için + sembolüyle **metin birleştirme (concatenation)** işlemi yapılır ("Merhaba " + "Öğretmenim" gibi). Akış şemalarında metin birleştirme ve atama adımları, birer düzenleme eylemi kabul edildiğinden **dikdörtgen (işlem)** sembolüyle gösterilir.

9. ÖRNEK

DİNAMİK KARŞILAMA VE METİN BİRLEŞTİRME

Kod-Asistan yazılımına yeni giriş yapan bir kullanıcının ismini dışarıdan girdi olarak alan, ardından bu ismin başına kurumsal bir "Hoş Geldiniz" selamlaması ekleyerek birleştiren ve sonucu ekrana yazdıran algoritmanın akış şeması tasarımı Şema 1.7'de modellenmiştir.



Şema 1.7: Metin birleştirme ve karşılama algoritmasının akış şeması

1.2.4. Dijital Depo

Yazılım geliştirme sürecinde algoritmaları sözde kod ve akış şemaları ile modellemek, yazılım geliştiriciyi büyük mantık hatalarından korur. Profesyonel yazılım dünyasında kod yazmaya başlamadan önce projenin tüm tasarım belgelerinin, algoritmalarının, sürümlerinin ve amacının güvenli bir şekilde saklandığı dijital bir alana ihtiyaç duyulur. Bu alana **dijital depo (repository)** adı verilir. Yazılım projelerinin zamanla büyümesiyle sürüm (versiyon) numaraları büyür. Sürüm numaraları v0.1, v1.0 gibi gelişim sürüm adlarıyla takip edilir. Bir dijital deponun en önemli belgesi ise projenin vitrini ve teknik imzası sayılan **README.md** dosyasıdır.

Profesyonel yazılım dünyasında hiçbir proje tek bir adımda hatasız ve mükemmel (v1.0) olarak tamamlanamaz. Süreçler her zaman planlı ve hiyerarşik bir tasarımla (v0.1) başlar; hatalar elendikçe, testler yapıldıkça ve yeni modüller eklendikçe sabırla olgunlaşır. Geliştirilen Kod-Asistan projesini bulut deposunda adım adım saklamak ve README vitrinini bu gelişime uygun olarak güncel tutmak, profesyonel bir yazılımcının projesine duyduğu özeni ve mesleki disiplinini yansıtan en somut göstergedir.

Dijital depolarda projeyi anlatan bu vitrin metinlerini biçimlendirmek (başlık yapmak, kalın yazmak veya listeler oluşturmak) için karmaşık kodlar yerine basit sembollerin (örneğin ana başlık için #, maddeler için * veya -) kullanıldığı pratik işaretleme diline **Markdown** adı verilir.

6. UYGULAMA

Uygulamanın Adı: DİJİTAL DEPO VE PROJE VİTRİNİNİN TASARLANMASI (KOD-ASİSTAN v0.1)

Uygulamanın Süresi: 40 dakika

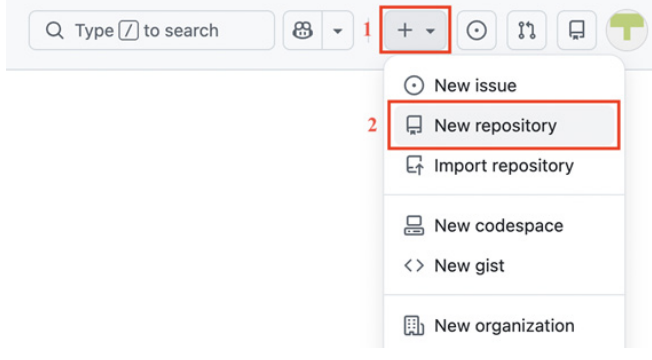
Uygulamanın Amacı: Sürüm kontrol sistemlerinin mantığını kavrayarak projeyi dijital depoda yapılandırmak, süreci emeğe saygı ve dürüstlük ilkelerine uygun şekilde belgelemek ve bu alandaki mesleki sorumlulukları eksiksiz yerine getirmek

Problem Senaryosu: Yazılım dünyasında projeler zamanla büyür, güncellenir ve yeni sürümler (v0.1, v1.0 vb.) kazanır. Bir yazılım geliştiricinin en temel sorumluluklarından biri, ürettiği projeyi diğer insanların kolayca anlayabileceği standartlarda, zamanında ve eksiksiz bir şekilde belgelemektir. Bu sorumluluğun bir gereği olarak Kod-Asistan projesinin tasarım belgelerinin, akış şemalarının ve mantıksal sürüm notlarının güvenli bir şekilde saklanacağı, projenin kimliği ve vitrini niteliğinde olan dijital bir alanın oluşturulması beklenmektedir. Projeyi inceleyecek diğer yazılımcıların koda bakmadan önce sistemin niteliğini ve geliştiricinin meslek etiğine olan bağlılığını anlaması, bu dijital vitrinin standartlara uygun ve dürüst bir şekilde hazırlanmasına bağlıdır.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. **Aşama: Dijital Deponun (Repository) Oluşturulması**
   - **a)** Tarayıcınız üzerinden küresel yazılım havuzu olan https://github.com/ platformunu açınız.
   - **b)** Açılan pencerenin sağ üst köşesindeki **Sign Up** butonuna basarak gelen kayıt formunu doldurunuz ve kayıt işleminizi tamamlayınız.
   - **c)** Hesabınızın sağ üst bölümünde yer alan + sembolüne, ardından **New repository (Yeni depo)** seçeneğine tıklayınız (Görsel 1.3).

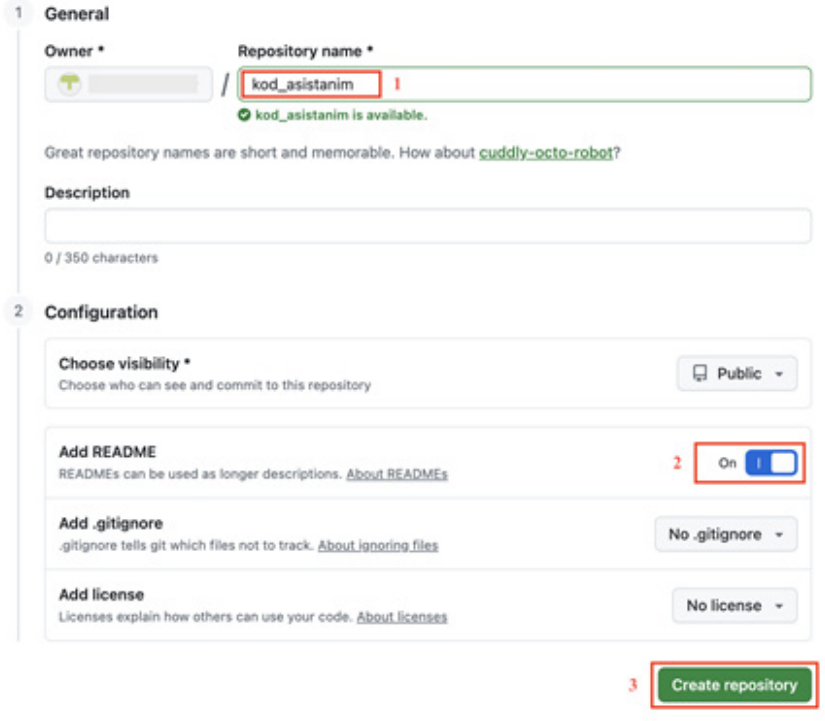


Görsel 1.3: Yeni bir depo oluşturma aşaması

   - **ç)** Açılan konfigürasyon ekranında **Repository name** alanına projenizin adı olan **kod_asistanim** ifadesini küçük harflerle yazınız.

6. UYGULAMA

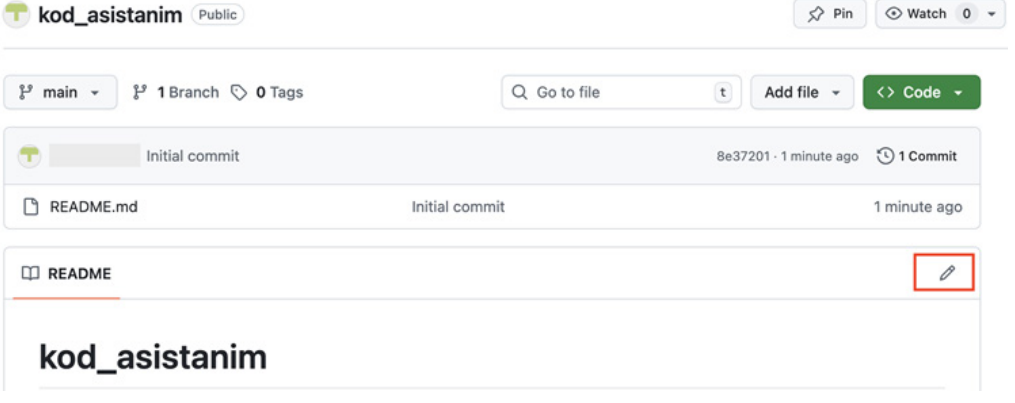
d) Deponun genel görünürlüğünü **Public** olarak belirleyiniz ve projenin ana belgesi olan **Add README** seçeneğini **On (Aktif)** durumuna getirerek genel deponuzu oluşturunuz (Görsel 1.4).



Görsel 1.4: Genel depo oluşturma işlemi

2. Aşama: Proje Vitrininin (README.md) Markdown ile Düzenlenmesi

a) kod_asistanim deponuzda bulunan kalem (düzenle) simgesine (Görsel 1.5) tıklayarak Markdown editörünü açınız.



Görsel 1.5: Markdown editörünü açma

6. UYGULAMA

b) Belgenizi şu standart başlık yapılarına göre doldurunuz:

Proje Yazarı: Adınızı ve soyadınızı yazınız.

 # Projenin Adı: Kod-Asistan v0.1

Projenin Amacı: Kullanıcılara günlük işlerinde (hesaplama, selamlama vb.) yardımcı olacak bir dijital asistan tasarlamak

Projenin Hedefleri: Bu asistanın ilerleyen zamanlarda hangi özellikleri kazanacağını (örneğin v1.0 Akıllı Menü, v2.0 Veri Saklama) maddeler hâlinde yazınız.

c) Projenin kapsam bölümüne daha önce tasarladığınız **KRİTİK EŞİK** sözde kod mantığını metinsel olarak yerleştiriniz.

ç) Ahilik kültüründeki ustaya saygı ve doğruluk ilkesinin modern yazılımdaki karşılığı olarak belgenin en altına yararlandığınız kaynakları (örneğin ders kitabınızı) içeren kurumsal bir **## Kaynakça** bölümü ekleyip atıflarınızı tamamlayınız.

Değerlendirme: Çalışmanız karekodda yer alan Gözlem Formu'yla değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Gözlem Formu'na ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Yazılım projelerinin dijital vitrini olan README belgeleri sadece metinlerden oluşmaz. Karmaşık algoritmaların, yazılım mimarilerinin ve sistem gereksinimlerinin diğer geliştiriciler tarafından hızlıca kavranabilmesi için dokümantasyonların görsel materyallerle desteklenmesi evrensel bir standarttır. Profesyonel bir yazılım dokümantasyonu; projenin çalışma anını gösteren ekran görüntülerini, Girdi-İşlem-Çıktı ilişkilerini özetleyen akış şemalarını ve sistemin mantıksal haritasını mutlaka içermelidir.

Dijital depolarda bu görselleri belgeye entegre etmek için Markdown işaretleme dilinin pratik yapısından faydalanılır. Projenin çalışma mantığını yansıtan bir akış şeması dijital depoya yüklendikten sonra, standart bir Markdown söz dizimi olan ![Şema Açıklaması](dosya_adi.png) yapısıyla README dosyasına kolayca yerleştirilir. Bu görselleştirme pratiği, projenin okunabilirliğini en üst düzeye çıkarırken yazılım geliştiricinin küresel sektör standartlarına olan teknik bağlılığını da kanıtlar. Geliştirilen her yeni algoritmanın standart akış şemasını ve örnek çalışma ekranını güncel dokümantasyona görsel olarak eklemek, yüksek nitelikli projeler üretmenin en temel kuralıdır.

7. UYGULAMA

Uygulamanın Adı: KOD-ASİSTAN BASİT KAYIT AKIŞI

Uygulamanın Süresi: 20 dakika

Uygulamanın Amacı: Kullanıcıdan alınan birden fazla farklı metinsel veriyi (ad, soyadı, kullanıcı adı vb.) mantıksal bir sıra dâhilinde işlemek, metin birleştirme operatörlerini kullanmak ve sonucu standart akış şeması sembolleriyle tasarlamak

Problem Senaryosu: Kod-Asistan projesinin kullanıcı yönetim modülü tasarlanacaktır. Asistanın kullanıcıya ismiyle hitap edebilmesi ve sistemi kişiselleştirebilmesi için kayıt aşamasında ad ve soyadı verilerinin ayrı ayrı alınması, arka planda birleştirilerek tek bir profil kimliği hâline getirilmesi gerekmektedir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulamanın Aşamaları

a) Senaryoda verilen beş adımlı (Başlama, İsim İsteme, Metin Birleştirme, Selamlama Mesajı Gösterme, Durma) doğrusal algoritmayı dikkatlice analiz ediniz.

b) Dijital çizim araçlarını kullanarak başlangıç / bitiş için oval, veri giriş / çıkış adımları için paralelkenar ve metin birleştirme işlemi için dikdörtgen sembollerini eksiksiz şekilde yerleştiriniz.

c) İşlem adımları arasındaki ilerleme hiyerarşisini belirten akış yön oklarını yukarıdan aşağıya doğru hatasız bir şekilde birbirine bağlayınız.

ç) Hazırladığınız dijital çizimi uygun formatta dışa aktararak (export) oluşturduğunuz kod_asistanim bulut deposuna yükleyiniz.

Değerlendirme: Yapacağınız çalışma, karekodda yer alan Dereceli Puanlama Anahtarı'na göre değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Kendi öğrenme sürecinizi değerlendirmek için diğer karekodda yer alan Öz Değerlendirme Formu'nu doldurunuz.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Öz Değerlendirme Formuna ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

3. SIRA SİZDE

1. Algoritmik Düşünme ve Tasarım öğrenme birimi boyunca gerçekleştirdiğiniz tüm uygulamaları, tasarım süreçlerinizi ve Kod-Asistan çalışmalarınızı gözünüzün önünden geçiriniz.
2. Aşağıda üç farklı geometrik sembolle tanımlanan alanlardaki (Üçgen, Kare, Çember) yönlendirme sorularını dikkatlice okuyunuz.
3. Öğrenme birimi konularıyla ilgili öğrendiklerinizden hareketle kendi öğrenme yolculuğunuzu yansıtan içten ve özgün cevaplarınızı verilen boşluklara yazınız.

ÜÇGEN: Zihnimde Şekillenen 3 Temel Kazanım

Bu öğrenme biriminin sonunda artık tam anlamıyla uzmanlaştığımı düşündüğüm, başkasına hiç zorlanmadan anlatabileceğim **3 temel algoritmik beceri / bilgi** şunlardır:

1. .....
2. .....
3. .....

KARE: Zihnimi "Köşeye Sıkıştıran" ve Netleşmesi Gereken Pürüzler

Algoritma tasarlarken, akış şeması çizerken veya dijital depoda sürüm kontrolü yaparken beni hâlâ zorlayan, zihnimde tam olarak oturmayan ve öğretmenime sormam gereken köşeli / pürüzlü noktalar şunlardır:

- .....

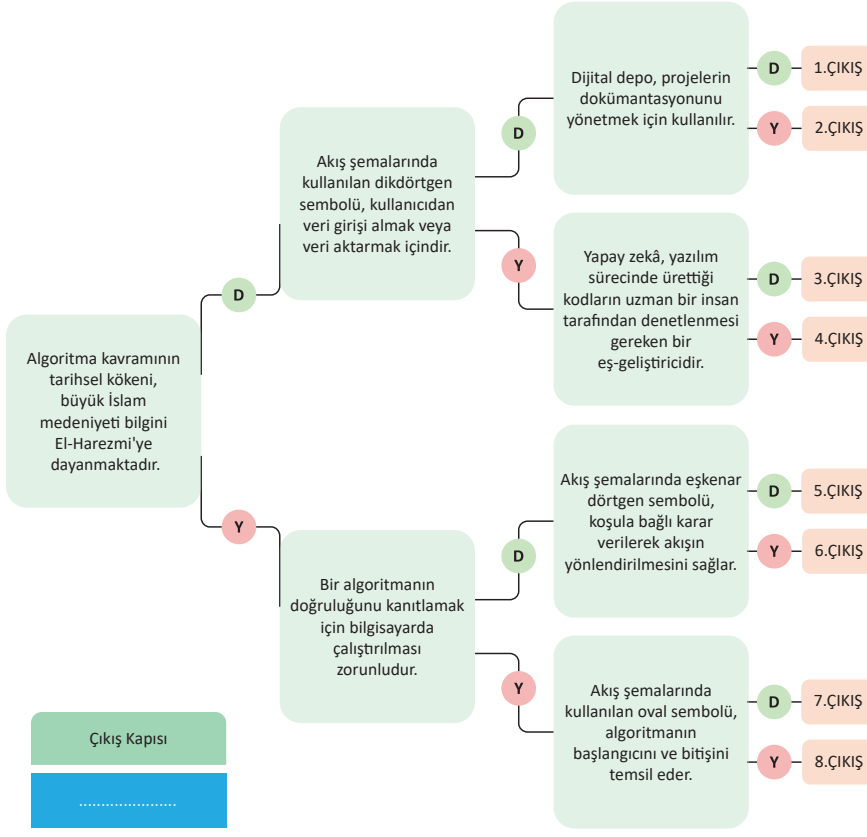
ÇEMBER: Zihnimde Dönen ve Hayatıma Değer Katan Etik Boyut

Yazılım dünyasında kod üretirken yapay zekâyı bir eş-geliştirici olarak denetlemenin önemi, Ahilik kültürü, meslek etiği ve emeğe saygı (README dosyasına kaynakça / atıf ekleme) hakkında **zihnimde yer eden ve hayatım boyunca dikkat edeceğim temel düşünce** şudur:

- .....

ÖLÇME VE DEĞERLENDİRME

A) Aşağıdaki Tanılayıcı Dallanmış Ağaç şemasında yer alan 1. ifadeden (kökten) başlayarak okuma yapınız. Okuduğunuz ifadenin doğru (D) veya yanlış (Y) olduğuna karar vererek ilgili ok yönünde (dallanarak) bir sonraki ifadeye ilerleyiniz. Her yol ayrımında verdiğiniz kararlar sizi en son aşamada 8 farklı çıkış kapısından yalnızca birine ulaştıracaktır. Kendi zihinsel mantık akışınızı takip ederek ulaştığınız çıkış kapısının numarasını sayfa altındaki kutucuğa işaretleyiniz.



B) Aşağıda 1'den 9'a kadar numaralandırılmış kutucuklardan oluşan yapılandırılmış grid tablosunu ve verilen soruları dikkatlice inceleyiniz. Her sorunun cevabı veya cevapları tablodaki kutucuklarda yer almaktadır. Bir kutucuğu birden fazla sorunun cevabı olarak kullanabilirsiniz. Soruların altındaki boşluklara, doğru olduğunu düşündüğünüz kutucuk numaralarını yazınız.

1. Girdi	2. Eş-Geliştirici	3. Sözde Kod
4. Eşkenar Dörtgen	5. README.md	6. Oval
7. Dikdörtgen	8. Akış Şeması	9. Çıktı

ÖLÇME VE DEĞERLENDİRME

a) Bir bilgisayar programının dış dünyayla veri alışverişi yaptığı (bilgi aldığı veya ekrana sonuç verdiği) süreçleri ifade eden kavramlar hangileridir?

.....

b) Tasarlanan bir algoritmanın mantıksal adımlarını bilgisayar kodlamasına geçmeden önce görsel veya metinsel olarak modellemek için kullanılan tasarım araçları hangileridir?

.....

c) Akış şemalarında süreci başlatmak, bitirmek, matematiksel hesaplama yapmak veya koşullu karar vermek amacıyla kullanılan evrensel geometrik semboller hangileridir?

.....

ç) Bugünkü yazılım dünyasında modern proje geliştirme süreçlerinde (profesyonel dokümantasyon hazırlama ve yapay zekâ kullanımı bağlamında) faydalanılan temel bileşenler hangileridir?

.....

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Aşağıdakilerden hangisi algoritmik düşünme sürecinde karmaşık bir problemin daha küçük ve yönetilebilir parçalara bölünmesini ifade eder?

A) Çıktı analizi
B) Görsel programlama
C) Problemi bileşenlerine ayırma
D) Sürüm kontrol yönetimi
E) Yapay zekâ kod denetimi

2. Akış şemalarında kullanılan oval sembolünün <u>temel</u> görevi nedir?

A) Çıktı mesajı yazdırmak
B) Karar hiyerarşisi kurmak
C) Matematiksel işlem yapmak
D) Süreci başlatmak veya bitirmek
E) Veri girişi sağlamak

3. Aşağıdakilerden hangisi bir öğrenci projesinde kullanıcıdan isim alma adımının GİÇ yapısındaki karşılığıdır?

A) Girdi
B) Çıktı
C) Döngü
D) İşlem
E) Karar

4. Yapay zekâ tarafından üretilen kodların bir uzman tarafından denetlenmesinin temel nedeni aşağıdakilerden hangisidir?

A) Kodun daha kısa sürede yazılmasını sağlamak
B) Mantıksal ve etik hataları saptamak
C) Bilgisayarın işlemci hızını artırmak
D) Programın grafik tasarımını otomatikleştirmek
E) Dosyaların daha az depolama alanı kullanmasını sağlamak

5. Aşağıdakilerden hangisi bir dijital depodaki README dosyasının profesyonel bir yazılımcı için anlamını ifade eder?

A) Kodların yedeklendiği gizli bir klasördür.
B) Projenin sadece bitiş tarihini gösterir.
C) Projenin teknik ve vizyoner imzasıdır.
D) Sadece kod hatalarının yazıldığı yerdir.
E) Yapay zekânın tüm kodlarını yazdığı alandır.

6. Aşağıdakilerden hangisi akış şemasında veri girişi ve veri çıkışı işlemleri için kullanılan ortak semboldür?

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

7. Kod yazmaya başlamadan önce çözüm planının netleştirilmesi işlemine ne ad verilir?

A) Algoritma tasarımı
B) Bellek yönetimi
C) Derleme
D) Hata ayıklama
E) Nesne yönelimli tasarım

8. Yazılım projelerinde kullanılan v0.1, v1.0, v2.0 gibi ifadeler aşağıdakilerden hangisini belirtir?

A) Bilgisayarın işlemci hızını
B) Projenin gelişim sürümlerini
C) Projenin kapladığı disk alanını
D) Toplam kullanıcı sayısını
E) Kod yazan kişi sayısını

ÖLÇME VE DEĞERLENDİRME

9. Aşağıdakilerden hangisi "kullanıcının girdiği yaş değeri 18'den küçük değilse" ifadesini temsil eden karşılaştırma operatörüdür?

A) <
B) ==
C) !=
D) >=
E) <=

10. Aşağıdakilerden hangisi bir algoritmanın kontrol ve test aşamasında dry run tekniğinin kullanılmasının temel amacıdır?

A) Kodları daha hızlı yazmak
B) Programın görselliğini artırmak
C) Verilerin internet üzerinden paylaşılmasını sağlamak
D) Algoritmadaki Girdi-İşlem-Çıktıyı tespit etmek
E) Yazılım diline geçmeden önce mantık hatalarını tespit etmek

Ç) Aşağıdaki soruların cevaplarını akademik bir dille ve teknik terimleri kullanarak açıklayınız.

1. Girdi-İşlem-Çıktı yapısını günlük hayattan bir problem (örneğin bankamatikten para çekme) üzerinden detaylandırarak açıklayınız.

.....
.....

2. Akış şeması kullanmanın sadece metin tabanlı bir algoritma yazmaya göre ne gibi avantajları vardır?

.....
.....

3. Yapay zekânın bir stajyer olarak tanımlanması durumunda bir yazılımcının yapay zekâ çıktısını neden doğrudan kopyalamaması gerektiğini etik ve teknik açıdan karşılaştırarak yazınız.

.....
.....

4. Profesyonel bir README.md dosyasında bulunması gereken temel başlıklar nelerdir ve bu dosyanın dijital portfolyo için önemi nedir?

.....
.....

5. Ahilik kültüründeki usta-çırak ilişkisi ve mesleki disiplinin, bir yazılım projesindeki sürüm kontrolü ve kod gözden geçirme süreçleriyle benzerliklerini açıklayınız.

.....
.....

ÖLÇME VE DEĞERLENDİRME

6. Aşağıdaki akış şeması, kullanıcıdan alınan üç farklı sayıdan (A, B, C) en büyüğünü bulup ekrana yazdırmak için tasarlanmıştır. Akış şemasındaki hatayı tespit edip düzeltiniz.

a) Akış Şeması ve Hata Tespiti: Akış şemasını dikkatlice inceleyerek tespit ettiğiniz mantıksal hatayı ve bu hatanın düzeltilmesi için yapılması gereken değişikliği gerekçesiyle yazınız.

……

……

b) Senaryo Analizi: Kullanıcının A=10, B=5, C=6 değerlerini girdiğini düşünerek aşağıdaki sorulara cevap veriniz.

- Şema hangi yolları izler?

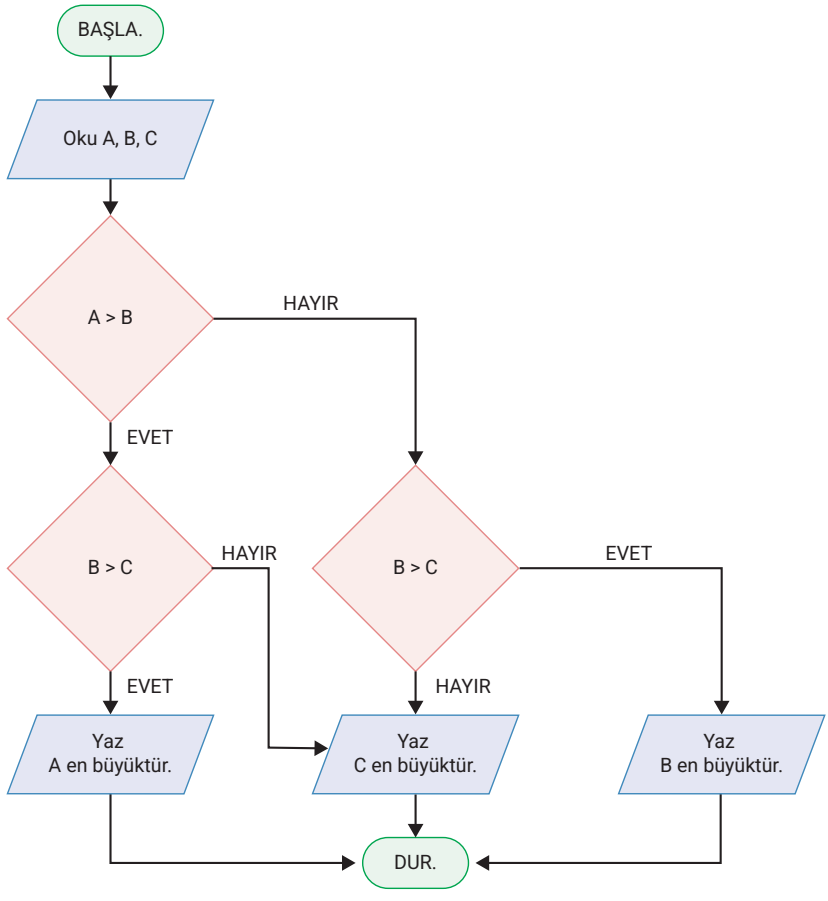
  ……

- Akış şeması sonuç olarak ekrana hangi mesajı yazdırır?

  ……

- Bu sonuç gerçekte en büyük sayı mıdır? Gerekçeleriyle açıklayınız.

  ……



ÖLÇME VE DEĞERLENDİRME

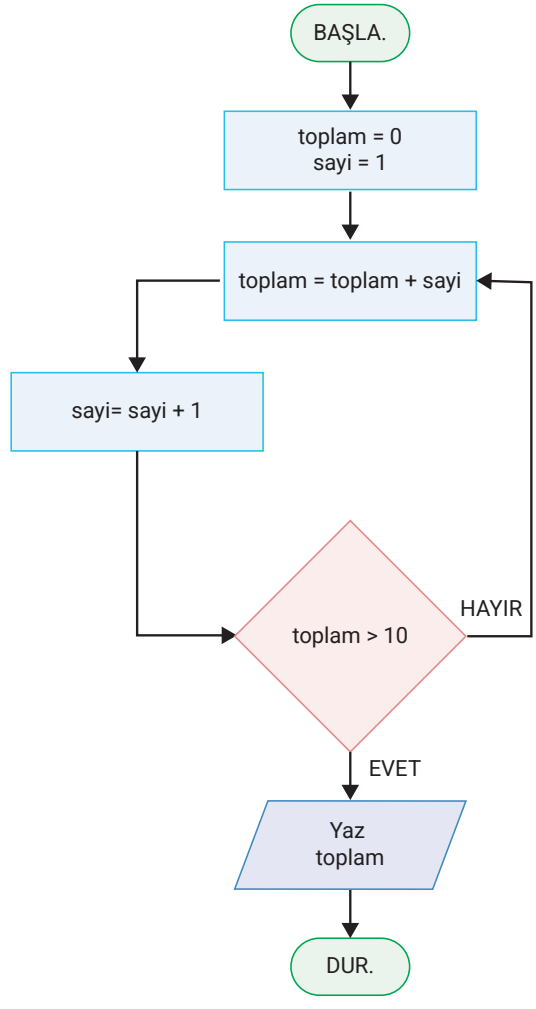
7. Verilen akış şemasını inceleyerek aşağıdaki sorulara cevap veriniz.

a) Algoritma bittiğinde ekrana yazdırılan TOPLAM değeri kaçtır?

.....

b) Döngü toplamda kaç kez dönmüştür? (İpucu: Bir kâğıda İz Sürme Tablosu çizerek değişkenleri takip ediniz.)

.....




Hello
World

Öğrenme Birimi Karekodu

2. ÖĞRENME BİRİMİ
TEMEL PROGRAMLAMA YAPILARI

KONULAR

2.1. DEĞİŞKENLER, SABİTLER VE VERİ TÜRLERİ
2.2. KULLANICI ETKİLEŞİMİ VE TÜR DÖNÜŞÜMÜ
2.3. OPERATÖRLER VE İŞLEM ÖNCELİĞİ
2.4. YAPAY ZEKÂ KOD DENETİMİ

ANAHTAR KAVRAMLAR

- açıklama satırı
- aritmetik operatörler
- değişken
- derleyici
- işlem önceliği
- karşılaştırma operatörleri
- mantıksal operatörler
- programlama dili
- sabit
- sürüm kontrolü
- tip dönüşümü
- yorumlayıcı

BEKLENTİLER

Bu öğrenme biriminde sizden

- değişkenleri ve sabitleri programlama standartlarına göre tanımlamanız
- giriş-çıkış fonksiyonlarını kullanarak kullanıcı etkileşimi sağlamanız
- operatörleri ve işlem önceliğini kullanarak verileri işlemeniz
- yapay zekâ tarafından üretilen kodları İnsan Denetimi prensibiyle doğrulamanız

beklenmektedir.

BAŞLARKEN

Kâğıt üzerinde çizilen bir harita ne kadar kusursuz olursa olsun, yola çıkılmadan hedefe ulaşılamaz. Önceki öğrenme biriminde karmaşık problemleri analiz etmeyi ve çözüm adımlarını akış şemalarıyla kâğıda dökmeyi öğrendiniz. Şimdi ise zihninizdeki bu soyut tasarımları, bilgisayarın anlayabileceği canlı ve etkileşimli bir yazılıma dönüştürme zamanı!

Bilgisayarlar, binlerce veriyi birbirine karıştırmadan belleğinde nasıl tutar? Yazdığınız bir program, klavyeden gelen komutlara nasıl tepki verir ve kullanıcıyla nasıl iletişim kurar? Bu öğrenme biriminde, yazılım dünyasının en popüler dillerinden biri olan Python ile tanışarak bu soruların cevaplarını uygulamalı şekilde keşfedeceksiniz. Dönem boyunca sarmal yapıda geliştirdiğiniz Kod-Asistan projeniz, bu aşamayla birlikte yalnızca bir kâğıt taslağı olmaktan çıkacak; sizinle doğrudan iletişim kurabilen, anlık verileri hafızasında saklayabilen ve temel kararlar alabilen dijital bir yardımcıya dönüşecektir.

HAZIRLIK ÇALIŞMALARI

1. İnsanların bir telefon numarasını hatırlama süreci ile bilgisayarların verileri bellekte adresler aracılığıyla saklayıp yeniden erişmesi arasında hangi benzerlikler ve farklılıklar vardır?
2. Kütüphanede kitapların farklı ölçütlere göre sınıflandırılması veriye erişimi nasıl kolaylaştırır? Benzer biçimde bilgisayarlarda metin, sayı ve mantıksal değerlerin farklı veri türleriyle temsil edilmesi, verilerin düzenli işlenmesine nasıl katkı sağlar?
3. Verilen anahtar kavramın size neleri çağrıştırdığını düşününüz. Bu kavramla ilgili aklınıza gelen en fazla 5 kelimeyi ilgili kutucuklara yazınız. Ardından bu kelimeleri kullanarak kavram hakkındaki ön bilgilerinizi yansıtan anlamlı bir cümle kurunuz.

Anahtar Kavram: PROGRAMLAMA

a)

b)

c)

ç)

d)

İlgili Cümlem:

.....

.....

.....

.....

.....



2.1. DEĞİŞKENLER, SABİTLER VE VERİ TÜRLERİ

Günlük hayatta program ve yazılım (software) kelimeleri sıklıkla birbirinin yerine kullanılsa da aralarında ince bir mühendislik sınırı bulunur. Örneğin bir aşçının mutfakta adım adım uyguladığı yemek tarifi aslında bir programdır. Bu tarifin yer aldığı kitabın tamamı, kullanılacak malzeme listeleri ve bu araçların nasıl yönetileceğini anlatan kullanım yönergeleri bir araya geldiğinde ise çok daha geniş bir ekosistem olan yazılım ortaya çıkar. Bilişim dünyasında da bilgisayarlara ve diğer dijital sistemlere belirli bir işi yaptırmak için verilen sıralı komutlar dizisi program olarak adlandırılırken; bu programların nasıl kullanılacağını anlatan belgeleri, kütüphaneleri ve verileri de kapsayan bütüncül yapıya yazılım denir. Bu karmaşık yazılım sistemlerinin ve programların inşa edilebilmesi için bilgisayarların anlayabileceği kurallı bir iletişim yöntemine ihtiyaç duyulur. İnsanların kendi aralarında anlaşmak için dili kullanması gibi geliştiricilerin de bilgisayarlara, cep telefonlarına, robotlara veya akıllı cihazlara belirli görevleri nasıl gerçekleştireceklerini iletmek için kullandıkları yapılandırılmış sisteme **programlama dili** adı verilir. Her programlama dilinin, konuşma dillerinde olduğu gibi kendine özgü kesin dil bilgisi (söz dizimi) kuralları bulunur.

Bilgisayar ve diğer elektronik sistemler; temelde sıfır ve birlerden (0 ve 1) oluşan, donanım mimarisine doğrudan hitap eden makine dili seviyesinde çalışır. Makine dili, işlemci tarafından doğrudan yürütülebilir ancak insanlar tarafından yazılması ve hata ayıklanması oldukça zordur. Bu durumda programcının asıl görevi devreye girer. Programcı; çözmek istediği problemi önce mantıksal adımlara bölerek bir algoritma oluşturur, bir başka deyişle problemi insan aklının kavrayabileceği bir düzene sokar. Ardından bu algoritmayı, insanlar tarafından yazılıp okunabilen ve çevirici programlar (derleyici veya yorumlayıcı) aracılığıyla makine diline dönüştürülebilen programlama dili aracılığıyla ifade eder. Bu anlamda programcı, problemi analiz eden ve çözüm üreten bir birey olarak bir tür çevirmen gibi çalışır. Programcı, insan düşüncesini ve çözüm mantığını makinenin anlayabileceği talimatlara dönüştürür. Programlama dilleri de bu çeviri sürecini mümkün kılan köprü görevini görür.

2.1.1. Programlama Dillerinin Seviyeleri

Tüm programlama dilleri aynı yapıda değildir. Bir programlama dilinin makine diline ne kadar yakın veya uzak olduğu onun **seviyesini** belirler. Programlama dilleri, donanıma yakınlık ve soyutlama düzeylerine göre düşük ve yüksek seviyeli olarak sınıflandırılır; C ve C++ gibi bazı diller ise her iki düzeyin özelliklerini taşıdıkları için kimi kaynaklarda orta seviyeli olarak adlandırılır.

Düşük seviyeli diller, makine diline çok yakın olan dillerdir. Bu dillerde yazılan komutlar doğrudan donanımı kontrol eder ve makine diline çevrilmesi çok az işlem gerektirir. Bu sayede programlar son derece hızlı ve verimli çalışır ancak yazılması ve anlaşılması oldukça zordur, programcının donanımı çok iyi tanıması gerekir. Assembly dili, bu kategorinin en bilinen örneğidir.

Orta seviyeli diller hem donanıma yakın esnek bir kontrol imkânı sunan hem de yüksek seviyeli dillerin okunabilirliğine yakın bir söz dizimi kullanan dillerdir. C ve C++ dilleri bu yaklaşıma örnek gösterilebilir.

Yüksek seviyeli diller ise insan diline çok daha yakın olan dillerdir. Günlük konuşma diline benzer bir sözdizimi kullandıklarından öğrenmesi ve yazması çok daha kolaydır. Programcı, donanımın teknik ayrıntılarıyla uğraşmak zorunda kalmadan çözüm odaklı düşünebilir. **Python**, **Java**, **C#** ve **JavaScript** bu kategoriye giren dillerden bazılarıdır. Yüksek seviyeli diller makine diline daha uzak olduğundan aralarındaki uçurumu kapatmak için **derleyici** veya **yorumlayıcıya** ihtiyaç duyarlar. Araya giren bu

çeviri ve soyutlama katmanları, yüksek seviyeli dillerin makine dillerine kıyasla nispeten daha yavaş çalışmasına neden olur.

Programlama dili seviyelerinin anlaşılırlık, hız, donanım kontrolü gibi kriterlere göre karşılaştırması **Tablo 2.1**'de özetlenmiştir.

Tablo 2.1: Programlama Dili Seviyelerinin Karşılaştırılması

Özellik	Düşük Seviyeli (Assembly)	Orta Seviyeli (C/C++)	Yüksek Seviyeli (Python)
Anlaşılırlık	Çok Zor	Orta	Çok Kolay
Soyutlama Düzeyi	Çok Düşük	Orta	Çok Yüksek
Donanım Kontrolü	Tam Kontrol	Yüksek Kontrol	Kısıtlı / Soyut
Taşınabilirlik	Donanıma Bağımlı (Mimariye Özel)	Taşınabilir	Çok Taşınabilir

2.1.2. Derleyiciler ve Yorumlayıcılar

Programlama diliyle yazılan talimatların makineye ulaşması için makine diline dönüştürülmesi gereklidir. Bu dönüşümü gerçekleştiren **derleyiciler** ve **yorumlayıcılar** olarak iki temel yaklaşım vardır.

Derleyici (compiler); programcının yazdığı kodun tamamını önce baştan sona okur, yazım hatalarını kontrol eder ve tüm kodu bir bütün olarak makine diline çevirir. Kodun yazımında (söz diziminde) bir hata varsa derleme işlemi tamamlanmaz ancak derleyicilerin mantık hatalarını her zaman fark edemeyeceği unutulmamalıdır. Çalışma mantığı olarak bu süreç, bir kitabın tamamının başka bir dile çevrilmesinden sonra okunmasına benzer. **C** ve **C++** bu yaklaşımı kullanan dillere örnek verilebilir.

Yorumlayıcı (interpreter) ise kodu bir bütün olarak değil satır satır okuyarak anında çalıştırır. Bir satır çevrilip çalıştırıldıktan sonra sıradaki satıra geçilir. Hata tespiti açısından yorumlayıcı, hatanın bulunduğu satıra gelene kadar programı çalıştırır ve o noktada durur. Bu yaklaşım tıpkı bir tercümanın, konuşmacının cümlelerini anlık olarak çevirmesine benzer. **Python** ve **Ruby**, bu yaklaşımı kullanan dillere örnek gösterilebilir. İki yaklaşım arasındaki temel fark **hız ve esnekliktir**. Derleyici ve yorumlayıcı yaklaşımlarına ait temel özelliklerin karşılaştırması Tablo 2.2'de verilmiştir.

Tablo 2.2: Derleyici ve Yorumlayıcı Yaklaşımlarının Karşılaştırılması

Özellik	Derleyici	Yorumlayıcı
Çalışma Mantığı	Kodun tamamını bir kerede makine diline çevirir ve bir dosya (.exe gibi) oluşturur.	Kodu satır satır okur ve o an çalıştırır.
Hız	Program bir kez derlendikten sonra çok hızlı çalışır.	Her seferinde satır satır okunduğu için nispeten daha yavaştır.
Hata Tespiti	Tüm kod hatasız olana kadar programı çalıştırmaz.	Hatanın olduğu satıra gelene kadar programı çalıştırır, hatayı görünce durur.
Örnek Diller	C, C++, Rust	Python, Ruby



Bugün birçok programlama dili hem derleme hem yorumlama tekniklerini birlikte kullanabilir. Örneğin Java ve C# önce ara bir dile derlenir, ardından sanal makine tarafından çalıştırılır.

2.1.3. Python Dili

Python; yüksek seviyeli, yorumlanabilen, geniş kütüphane desteğine sahip ve nesne yönelimli bir programlama dilidir. 1991 yılında Guido van Rossum tarafından geliştirilen bu dil, ismini geliştiricisinin ilgiyle takip ettiği **Monty Python's Flying Circus** adlı komedi grubundan almıştır.

Python dili; sade söz dizimi ve okunabilirlik avantajı sayesinde web geliştirmeden veri analizine, robotik kontrolden yapay zekâ mimarilerine kadar oldukça geniş bir alanda tercih edilir. Bu esnekliği ve güçlü yapısı nedeniyle Python; küresel yazılım ekosisteminde olduğu gibi Türkiye'nin yerli girişimlerinde, akıllı otomasyon sistemlerinde ve çeşitli teknoloji projelerinde etkin olarak kullanılır.

OKU-YORUM

Cezeri'nin Çarkından Python'un Kodlarına

Bugün bilgisayarların kalbinde yer alan programlama mantığı, köklü bir matematik ve mekanik mirasına dayanır. 12. yüzyılda Artuklu Sarayı'nda yaşayan **Ebû'l İz El-Cezeri**, yazdığı **Kitâb-ül Hiyel (Olağanüstü Mekanik Araçların Bilgisi Hakkında Kitap)** adlı eseriyle kontrol ve otomasyon tarihinin önemli örnekleri arasında değerlendirilir. Düzeneklerdeki ardışık hareketler ve mekanik kontrol noktaları, bugünün algoritmalarındaki sıralı işlem ve koşul yapılarıyla kavramsal benzerlikler taşır.

Bilim tarihindeki köklü matematiksel miras ve Cezeri ile somutlaşan otomasyon dehası, bugün Python gibi modern diller aracılığıyla dijital bir boyuta taşınmıştır. **Millî Teknoloji Hamlesi** kapsamında siber güvenlik ve yazılım projelerini hayata geçiren genç geliştiriciler, bu köklü mantıksal mirası devam ettirmektedir. Python öğrenmek ve algoritmik tasarımlar yapmak, Cezeri'nin mekanik çarklarındaki işlem basamaklarını bugünün kod satırlarına aktararak geleceğin teknolojisini inşa etmek anlamına gelir.

Düşünelim-Tartışalım

Okuduğunuz metinden hareketle Cezeri'nin mekanik tasarımları ile modern Python kodları arasındaki mantıksal köprüyü kurarak tablodaki ilgili boşluklara Python programlamasındaki karşılıklarını yazınız.

Cezeri'nin Mekanik Tasarımları	Python Programlamasındaki Karşılığı
Belirli bir düzende sürekli ve ardışık olarak dönen dişli çarklar	
Suyun akış yönünü veya miktarını şarta bağlayan mekanik kontrol noktaları	
Mekanizmanın başından sonuna kadar adım adım nasıl çalışacağını gösteren mühendislik planı	

1. UYGULAMA

Uygulamanın Adı: PYTHON KURULUMU

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: Python programlama dilini resmî kaynaklardan güvenle indirmek, bilgisayarınıza kurmak ve komut satırı üzerinden çalıştığını test etmek

Problem Senaryosu: Sarmal yapıda geliştirilen Kod-Asistan projesinin tasarım aşaması (v0.1) tamamlanarak kodlamaya geçilmiştir. Komutların bilgisayarda yürütülebilmesi için sisteme Python yorumlayıcısının kurulması zorunludur. Güvenli ve kararlı bir geliştirme ortamının inşası, bu kurulumun standartlara uygun gerçekleştirilmesine bağlıdır.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Güvenli Kaynaktan İndirme

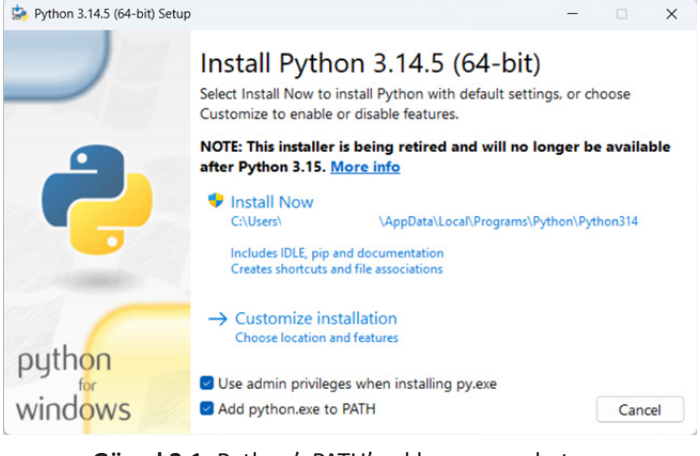
a) İnternet tarayıcısı üzerinden Python programlama dilinin resmî ve güvenilir dağıtım kaynağı olan https://www.python.org/ adresine erişim sağlayınız.

b) Ana sayfada yer alan üst menüdeki Downloads sekmesinin üzerine gelince açılan panelden, işletim sistemi mimarisine uygun olan güncel Python 3.x.x sürüm bağlantısına tıklayarak kurulum dosyasını yerel depoya indiriniz.

2. Aşama: Ortam Değişkenlerinin Yapılandırılması ve Kurulum

a) İndirilen çalıştırılabilir kurulum dosyasını çalıştırarak kurulum arayüzünü başlatınız.

b) Kurulum arayüzünün alt kısmında yer alan **Add python.exe to PATH (Python'ı PATH'e ekle)** onay kutucuğunu işaretleyiniz (Görsel 2.1). Bu seçeneğin işaretlenmesi, Python yorumlayıcısına komut satırından farklı dizinlerde doğrudan erişilebilmesini kolaylaştırdığı için önerilir.



Görsel 2.1: Python'ı PATH'e ekleme onay kutusu



1. UYGULAMA



c) Onay kutucuğunu işaretledikten sonra üst bölümde bulunan Install Now (Şimdi Kur) butonuna tıklayarak yükleme sürecini başlatınız.

ç) Ekranda Setup was successful (Kurulum başarılı) ifadesi görüldüğünde Python yorumlayıcısının sisteme kararlı bir şekilde entegre edildiği anlaşılır. **Close (Kapat)** butonuna tıklayarak kurulum penceresini sonlandırınız.

3. Aşama: Kurulumun Komut Satırından Test Edilmesi

a) Kurulumun doğrulanması ve sistem değişkenlerinin kararlılığının kontrol edilmesi amacıyla işletim sisteminin arama çubuğuna cmd yazarak Komut İstemi (Terminal) uygulamasını açınız.

b) Açılan komut satırı ekranına python --version komutunu yazıp klavyeden **Enter** tuşuna basınız.

c) Komut çalıştırıldıktan sonra alt satırda Python 3.x.x (Görsel 2.2) şeklinde yüklenen güncel sürüm numarası ekrana yansıyorsa Python geliştirme ortamının sorunsuz bir şekilde hazırlandığı onaylanmış olur.

```
C:\>python --version
Python 3.13.5
```

Görsel 2.2: Python güncel sürüm numarası

2.1.4. Python Interactive Shell ve Idle

Python yorumlayıcısının bilgisayara başarılı bir şekilde entegre edilmesinin ardından kod satırlarının yazılacağı ve saniyeler içinde test edileceği yazılım ortamının seçilmesi gerekir. Python, kurulumuyla birlikte kullanıcıya iki temel çalışma alanı sunar.

Etkileşimli Kabuk (Interactive Shell): Komut satırı üzerinden doğrudan python yazılarak ulaşılan bu alan, yazılan her bir kod satırını Enter tuşuna basıldığı anda anlık olarak çalıştırır ve sonucu alt satıra yansıtır. Büyük projeler geliştirmek için uygun olmasa da küçük kod parçalarını hızla test etmek ve yorumlayıcının davranışını incelemek açısından oldukça işlevseldir.

IDLE [Integrated Development and Learning Environment (Entegre Geliştirme ve Öğrenme Ortamı)]: Python kurulumuyla birlikte yerleşik olarak gelen basit ve hafif bir **IDE [Integrated Development Environment (Entegre Geliştirme Ortamı)]** yazılımıdır. Sıklıkla birbirine karıştırılan bu iki kavramdan IDE, yazılım geliştirmek için kullanılan tüm araçların (editör, derleyici vb.) genel kategori adıyken, IDLE sadece Python'a özel olarak eğitim amacıyla tasarlanmış spesifik bir programın adıdır. IDLE, programcılara hem etkileşimli kabuk gibi anlık denemeler yapma imkânı sunar hem de .py uzantılı bağımsız kod dosyaları oluşturarak karmaşık algoritmaları kaydetme, düzenleme ve hata ayıklama kolaylığı sağlar.

2. UYGULAMA

Uygulamanın Adı: PYTHON IDLE İLE BASİT TOPLAMA İŞLEMİ

Uygulamanın Süresi: 20 dakika

Uygulamanın Amacı: IDLE üzerinde etkileşimli kabuk ve kod düzenleyici araçlarını kullanma, temel aritmetik operatörlerle matematiksel süreçleri kodlama ve program çıktılarını anlık olarak doğrulamak

Problem Senaryosu: Yerel sisteme kararlı bir şekilde entegre edilen Python yorumlayıcısının, tasarım aşaması (v0.1) tamamlanan **Kod-Asistan** projesinin ilk matematiksel fonksiyonlarını çalıştırması planlanmaktadır. Bu doğrultuda asistanın arka planda yürüteceği en temel veri işleme adımı olan **iki sayısal veriyi toplama ve sonucu ekrana yansıtma** sürecinin, IDLE araçları kullanılarak bilgisayar belleğinde somut şekilde çalıştırılması gerekmektedir.

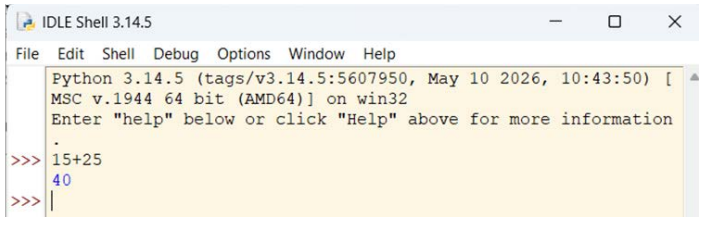
Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: IDLE Etkileşimli Kabuk Kullanımı

a) İşletim sisteminin başlat menüsünden IDLE (Python 3.x.x) uygulamasını arayarak açınız.

b) Ekrana gelen etkileşimli kabuk alanına Python söz dizimine uygun temel matematiksel ifadeler (örneğin 15 + 25) yazarak anlık sonuçları inceleyiniz (Görsel 2.3).



Görsel 2.3: IDLE matematiksel işlem sonucu

2. Aşama: Editör Modunda Program Dosyası Oluşturma

a) Etkileşimli kabukta yapılan denemeler doğrudan düzenlenebilir bir .py dosyası olarak saklanmadığı için bağımsız bir kaynak dosyası oluşturmak amacıyla IDLE üst menüsünden **File > New File** bağlantısına tıklayınız.

b) Açılan boş kod düzenleyici penceresine, iki sayının aritmetik toplama sürecini yürüten ve sonucu ekrana yazdıran şu standart Python komut bloklarını kodlayınız:

```
print(15+25)
```

c) Yazılan kod dosyasını yerel depoya kaydetmek amacıyla **File > Save As** menüsüne tıklayınız, dosya adını toplama_islemi şeklinde belirleyip .py (Python kaynak kodu) uzantısıyla kaydediniz.

2. UYGULAMA

3. Aşama: Kodun Yorumlayıcı Tarafından Çalıştırılması

a) Düzenleyici penceresinde kayıt altına alınan kod blokunu çalıştırmak için üst menüde yer alan **Run > Run Module** seçeneğine tıklayınız veya klavyeden doğrudan F5 fonksiyon tuşuna basınız.

b) Editör penceresinden ayrılan akış, otomatik olarak IDLE kabuk (shell) ekranına geçiş yapar ve ekranda hesaplama işleminin somut çıktısı olan 40 değerini yansıtır. Böylece kâğıt üzerinde tasarlanan toplama algoritması, bilgisayar ortamında başarılı bir şekilde çalıştırılır (Görsel 2.4).

```
toplama_islemi.py
40
```

Görsel 2.4: IDLE'de program çıktısının görüntülenmesi

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme formuna ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

2.1.5. Yazılım Geliştirme Ortamları

Python yerleşik etkileşimli kabuğu, programlama süreçlerinin temel mantığını kavramak ve küçük kod parçalarını anlık olarak test etmek açısından işlevsel bir yapı sunar. Bununla birlikte geliştirilen projelerin hacmi büyüdükçe çoklu dosyaların yönetilmesi, yazım hatalarının anlık olarak denetlenmesi ve kod bloklarının kararlı bir hiyerarşiyle yapılandırılması adına daha gelişmiş yazılım araçlarına ihtiyaç duyulur. Yazılım dünyasında bu amaçla kullanılan araçlar temel olarak **Kaynak Kod Editörleri** ve **Entegre Geliştirme Ortamları** olmak üzere iki kategoride incelenir. Kaynak kod editörü ve IDE kavramlarına ait temel karşılaştırmalar Tablo 2.3'te verilmiştir.

Tablo 2.3: Kaynak Kod Editörü ve IDE Karşılaştırması

Özellik	Kaynak Kod Editörleri	Entegre Geliştirme Ortamları (IDE)
Temel İşlev	Metin düzenleme ve söz dizimi kurallarına göre kod renklendirmedir.	Yazılımın kodlanması, test edilmesi, hata ayıklanması ve derlenmesini içerir.
Yapı ve Hız	Temelde hafif ve çok hızlı çalışan metin editörleridir.	Çok sayıda aracı tek çatı altında topladığı için daha kapsamlıdır.
Genişletilebilirlik	Haricî eklentilerle (plugin) desteklenerek güçlendirilir.	Gerekli tüm araçlar ve kütüphane yöneticileri sisteme yerleşik gelir.

Gelişmiş kod editörleri ve IDE'ler, yazılım geliştirme süreçlerini şu özelliklerle destekler:

Akıllı Kod Tamamlama: Yazım esnasında komutların eksik kalan kısımlarını otomatik şekilde önererek kodlama hızını artırır ve hatalı komut yazılma riskini azaltır.

Söz Dizimi Denetimi (Syntax Checking): Kodlama esnasında yapılan yazım hatalarını ve eksik bırakılan karakterleri (parantez, tırnak işareti vb.) anında tespit ederek yazılımcıyı uyarır.

Kod Renklendirme (Syntax Highlighting): Kaynak kod içerisindeki anahtar kelimeleri, değişkenleri, metinsel ifadeleri ve sayısal değerleri farklı renk tonlarıyla yansıtarak kodun okunabilirliğini ve takibini kolaylaştırır.

Proje ve Dosya Yönetimi: Birden fazla kaynak kod dosyasının, tasarım belgesinin ve sürüm notunun tek bir panel üzerinden düzenli ve hiyerarşik bir şekilde yönetilmesini sağlar.

Yazılım projelerinin ihtiyaçlarına, seviyelerine ve uygulama alanlarına göre tercih edilen popüler Python geliştirme ortamları Tablo 2.4'te listelenmiştir.

Tablo 2.4: Popüler Python Geliştirme Ortamları ve Kullanım Alanları

Geliştirme Ortamı	Türü	Öne Çıkan Özellikleri ve Kullanım Alanları
Visual Studio Code	Kaynak Kod Editörü	Hafif ve esnek yapısıyla öne çıkar. Resmî Python eklentileri vasıtasıyla IDE özelliklerini kazanarak kararlı bir çalışma alanı sunar.
PyCharm (Community)	IDE	Python dili için özel tasarlanmıştır. Proje yönetimi, gelişmiş hata ayıklama ve kod analiz araçları bakımından yapılandırılmış bir mimariye sahiptir.
Jupyter Notebook	Web Tabanlı Ortam	Kodların bağımsız hücreler hâlinde adım adım çalıştırılmasına ve sonuçların anlık görselleştirilmesine imkân tanır. Özellikle veri analizinde tercih edilir.
Spyder	IDE	Bilimsel hesaplama araçlarını tek bir arayüzde toplar. Matematiksel modellemeler ve akademik araştırmalar için optimize edilmiştir.
Google Colab	Bulut Tabanlı Ortam	Yerel kurulum gerektirmeyen tarayıcı tabanlı bir ortamdır. Uzak sunucular üzerinden Python kodları yazılmasına ve çalıştırılmasına olanak tanır.

Bilgi Kutusu

BULUT TABANLI KODLAMA ALTERNATİFİ

Hiçbir program kurmadan, sadece bir internet tarayıcısıyla güçlü sunuculara bağlanıp kod yazabileceğinizi biliyor muydunuz? Programlama dünyasına adım atarken bilgisayara program kurmakta zorluk yaşanması veya laboratuvar bilgisayarlarında yetki kısıtlaması bulunması artık bir engel değildir. Bu öğrenme birimindeki tüm Python kodları; hiçbir kurulum yapılmadan, bulut tabanlı platformlar kullanılarak doğrudan internet tarayıcısı üzerinden çalıştırılabilmektedir.

3. UYGULAMA

Uygulamanın Adı: GELİŞMİŞ KAYNAK KOD EDİTÖRÜNÜN KURULUMU VE YAPILANDIRILMASI

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: Visual Studio Code (VS Code) programını resmî kaynaklardan güvenle indirmek, bilgisayara kurmak ve Python çalışma alanını (Workspace) hazırlamak

Problem Senaryosu: Yerel sisteme entegre edilen Python yorumlayıcısının, sarmal olarak yürütülen **Kod-Asistan** projesinin kaynak kodlarını ve tasarım belgelerini daha düzenli işleyebilmesi istenmektedir. IDLE kabuk arayüzü temel denemeler için işlevsel olsa da projenin hacmi büyüdükçe çoklu dosyaların yönetilmesi ve akıllı kod tamamlama özelliklerinin kullanılması bir ihtiyaç hâline gelmiştir. Bu ihtiyacı karşılamak üzere endüstri standartlarında bir geliştirme ortamının yerel bilgisayarda kurulması ve yapılandırılması gerekmektedir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Güvenli Dağıtım Kanalından İndirme

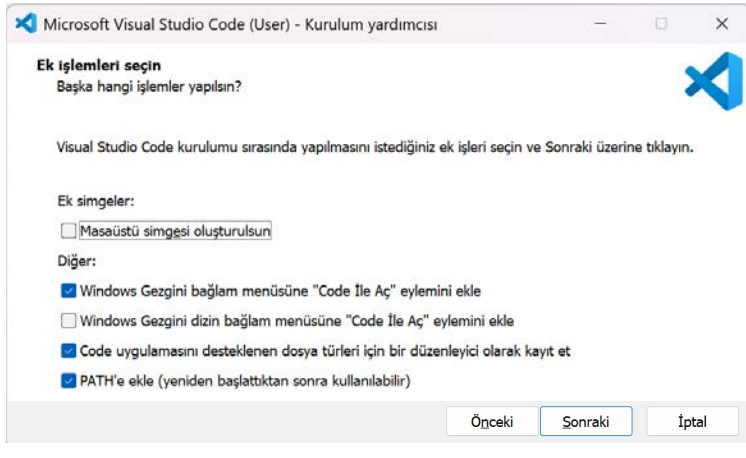
a) İnternet tarayıcısı üzerinden Visual Studio Code platformunun resmî dağıtım kanalı olan code.visualstudio.com adresine erişim sağlayınız.

b) Ana sayfada yer alan ve işletim sisteminin mimari yapısına uygun olarak sunulan indirme bağlantısına tıklayarak kurulum paketini yerel depoya aktarınız.

2. Aşama: Kurulum ve Sistem Değişkenlerinin Yapılandırılması

a) İndirilen çalıştırılabilir kurulum dosyasını çalıştırarak yükleme sihirbazını başlatınız. İlk aşamada ekrana gelen kullanıcı lisans sözleşmesi metnini inceleyerek, **Anlaşmayı kabul ediyorum** seçeneğini işaretleyip **Sonraki** butonuna tıklayınız.

b) Kurulum seçeneklerinin yer aldığı ekranda, işletim sisteminin komut satırı üzerinden editöre doğrudan erişebilmesini sağlayan **PATH'e ekle (add to PATH)** onay kutucuğunu işaretleyiniz. İsteğe bağlı olarak masaüstü simgesi oluşturma seçeneğini de seçerek **Sonraki** adımını takip ediniz (Görsel 2.5).



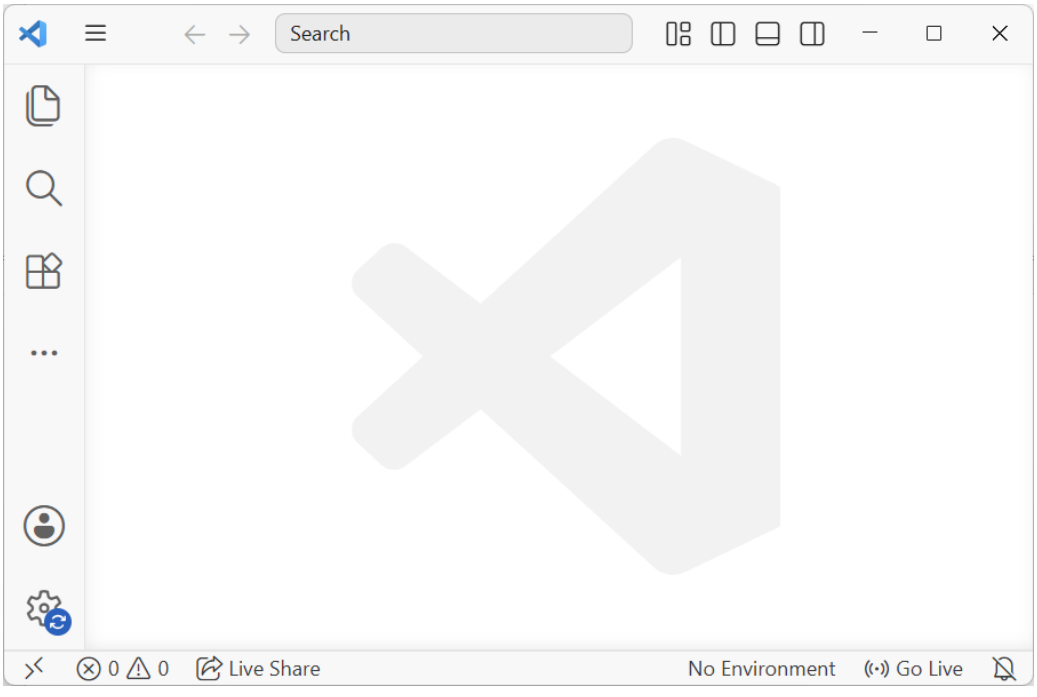
Görsel 2.5: VS Code ek işlemler penceresi

3. UYGULAMA

c) Özet penceresinin ardından **Kur** butonuna tıklayarak dosyaların sisteme aktarılma sürecini başlatınız. Yükleme barının tamamlanmasıyla birlikte editörü ilk kullanıma hazır hâle getiriniz.

3. Aşama: İlk Çalışma Alanının Oluşturulması

a) Program ilk açıldığında kullanıcıyı karşılayan sade arayüzün sol tarafında dikey bir araç paneli yer alır. Bu panel; dosya gezgini, arama modülü, kaynak kontrolü ve eklentiler pencerelerine hızlı erişim sağlar (Görsel 2.6).



Görsel 2.6: VS Code ekranı

b) Dönem boyu geliştirilecek Kod-Asistan projesine ait tüm belgelerin tek bir merkezden sistematik olarak yönetilebilmesi için üst menüde yer alan **File > Open Folder (Dosya > Klasör Aç)** sekmesi kullanılır. Bilgisayarın yerel deposunda önceden oluşturulmuş proje klasörünü seçerek editöre dâhil ediniz.

4. Aşama: Python Eklentisi ile Editörü Güçlendirme

a) Visual Studio Code'un ilk kurulduğunda genel amaçlı bir metin editörü yapısında olduğunu fark ediniz. Bu ortamın Python kaynak kodlarını algılayabilmesi, söz dizimi hatalarını anlık olarak denetleyebilmesi ve akıllı kod tamamlama mekanizmalarını devreye sokabilmesi için eklenti desteğiyle yapılandırılması gerektiğini göz önünde bulundurarak hazırlıklarınızı tamamlayınız.

3. UYGULAMA

b) Editör arayüzünün sol tarafında bulunan dikey araç panelindeki üst üste binen dört kare sembolüne sahip **Extensions (Eklentiler)** sekmesine tıklayınız (Bu panele klavyeden Ctrl + Shift + X kombinasyonuyla da doğrudan erişim sağlanabilir.).

c) Açılan eklenti mağazası arama çubuğuna aralarında boşluk bırakmadan Python ifadesini yazınız.

ç) Listelenen sonuçlar arasından yayıncısı Microsoft olan resmî Python eklentisini seçiniz. Eklenti detay panelinde yer alan mavi renkli **Install (Kur)** butonuna tıklayarak entegrasyon sürecini başlatınız.

d) Birkaç saniye süren indirme ve kurulum döngüsünün ardından butonun üzerinde **Uninstall (Kaldır)** yazısı belirdiğinde editörün Python kaynak kodlarını profesyonel standartlarda işleyebilecek teknik yetkinliğe ulaştığını doğrulayınız.

e) Editörün sağ üst köşesindeki **Run Python File (Oynat)** butonu vasıtasıyla Python dosyasını Terminal üzerinden doğrudan çalıştırınız.

5. Aşama: Projenin Dijital Depoya Aktarılması

Visual Studio Code üzerinde oluşturulan kod_asistanim çalışma alanının ve projenin başlangıç dosyalarının küresel bir bulut deposunda yedeklenmesi, kodların güvenliği ve dürüst sürüm kontrolü açısından kritik bir adımdır. Proje dosyalarınızı tarayıcı üzerinden GitHub platformuna yüklemek için aşağıdaki adımları planlı bir disiplinle uygulayınız.

a) İnternet tarayıcınızdan https://github.com adresine erişim sağlayıp giriş yapınız.

b) Kod asistan deponuzu açınız.

c) Depo ekranının üst menüsünde yer alan **uploading an existing file (mevcut bir dosyayı yükle)** bağlantısına tıklayınız.

ç) Bilgisayarınızın yerel deposundaki kod_asistanim klasörü içerisinde yer alan tüm başlangıç dosyalarını ve tasarım belgelerini sürükleyerek tarayıcıdaki bu alana bırakınız.

d) **Commit changes (Değişiklikleri kaydet)** kutucuğunun başlık alanına dürüstlük ilkesi çerçevesinde projenin ilk adımını tanımlayan **İlk sürüm yapılandırması tamamlandı** notunu yazınız ve **Commit changes** butonuna tıklayarak yükleme döngüsünü tamamlayınız.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme formuna ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

2.1.6. Temel Veri Türleri



Düşünelim-Tartışalım

Bir telefon numarasını aklınızda tutmaya çalışıyorsunuz ancak biri size başka bir şey söylediğinde veya dikkatiniz dağıldığında o numarayı hemen unutabiliyorsunuz. Bilgisayarların geçici belleği olan RAM (Rastgele Erişimli Bellek) de tıpkı insan zihninin bu kısa süreli hafızası gibi çalışır. Sistemin enerjisi kesildiğinde veya program kapatıldığında içindeki tüm geçici veriler (değişkenler) silinir. Sizce bilgisayarlar, işlem yaparken değişkenleri neden doğrudan kalıcı diske değil de RAM'e alırlar? Bu geçici varlığın çalışma hızına olan etkisi nedir? Düşüncelerinizi arkadaşlarınızla tartışınız.

Dikkat: input() fonksiyonu ile kullanıcıdan alınan veri, girilen karakterler sayı görünümünde olsa bile **metinsel (str)** türünde döndürülür. Sayısal işlemlerde kullanılabilmesi için int() veya float() gibi dönüşüm fonksiyonları uygulanmalıdır.

Bilgisayar belleğine kaydedilecek verilerin nitelikleri, boyutları ve üzerlerinde hangi matematiksel veya mantıksal işlemlerin yapılabileceği **veri türü (data type)** kavramı ile belirlenir. Python dilinde değişkenlerin veri türleri, türetildikleri anda dinamik olarak yorumlayıcı tarafından atanır.

Python, dinamik tipli (dynamically typed) bir programlama dilidir. Bu mimari gereği bir değişken tanımlanırken onun hangi veri türünde olacağını önceden belirtmek zorunlu değildir. Değişkenin veri türü, içine aktarılan değere göre otomatik olarak belirlenir. Daha da önemlisi, aynı değişken akış içerisinde farklı bir değer aldığında son atanan yeni değere göre veri türü de anlık ve dinamik şekilde değişir. Sektörel standartlara uygun olarak kullanılan dört temel veri türü şöyle yapılandırılmıştır:

Tam Sayı [Integer (int)]: Kesirli kısmı bulunmayan; pozitif, negatif veya sıfır değerini alabilen sayısal veri türüdür (örneğin ogrenci_sayisi = 34, hava_sicakligi = -5).

Ondalıklı Sayı [Floating Point (float)]: Kesirli veya ondalıklı kısımları bulunan, bilimsel hesaplamalarda ve hassas ölçümlerde kullanılan sayısal veri türüdür. Python dilinde ondalık kısımlar virgül (,) değil uluslararası standart gereği nokta (.) karakteriyle (örneğin urun_fiyati = 89.90, pi_sayisi = 3.14159) ayrılır.

Metinsel Veri [String (str)]: Tek tırnak '...' veya çift tırnak "..." işaretleri arasında tanımlanan; alfabetik karakterler, rakamlar ve özel sembollerden oluşan karakter dizileridir. Rakamlardan oluşsalar dahi string türündeki verilerle doğrudan aritmetik işlem (örneğin kullanici_adi = "gokturk_9", okul_numarasi = "1071") yapılamaz.

Mantıksal Veri [Boolean (bool)]: Sadece iki farklı durumun kontrolünde kullanılan, mantıksal karar süreçlerinin temelini oluşturan veri türüdür. Alabileceği değerler yalnızca **Doğru** veya **Yanlış** (örneğin sistem_acik_mi = True, bakiye_yeterli_mi = False) ifadeleridir.



2.1.7. type() Fonksiyonuyla Veri Türü Analizi

Yazılım geliştirme sürecinde bir değişkenin veya değerin veri türünü belirlemek için Python'un yerleşik type() fonksiyonu kullanılır. Bu fonksiyon, parametre olarak verilen değişkenin ya da değerin hangi veri türüne (sınıfa) ait olduğunu döndürerek programcının veri tipini doğrulamasını sağlar.

1. ÖRNEK

DİNAMİK VERİ TÜRÜ DEĞİŞİMİ VE DENETİMİ

Problem Senaryosu: Kod-Asistan projesinin merkezî veri yönetim katmanında asistan_verisi adında tek bir değişken tanımlanmıştır. Bu değişkenin program akışı boyunca aldığı yeni değerlere bağlı olarak bilgisayarın geçici belleğindeki yapısının nasıl dönüştüğü test edilecektir.

Örnek Kod Bloku

```
# 1. Aşama: Değişkene metinsel bir değer atanıyor.
asistan_verisi = "Kod-Asistan v0.2"
print(type(asistan_verisi))
# 2. Aşama: Aynı değişkene tam sayı türünde bir değer atanıyor (Veri türü değişir.).
asistan_verisi = 100
print(type(asistan_verisi))
# 3. Aşama: Aynı değişkene ondalıklı sayı türünde bir değer atanıyor (Veri türü değişir.).
asistan_verisi = 99.9
print(type(asistan_verisi))
# 4. Aşama: Aynı değişkene mantıksal bir değer atanıyor (Veri türü değişir.).
asistan_verisi = True
print(type(asistan_verisi))
```

Terminal Çıktı Analizi: Yazılan kod bloku çalıştırıldığında Python yorumlayıcısı asistan_verisi değişkenine her atamada yeni değeri atar. Bu nedenle type() fonksiyonu, değişkenin her adımda sahip olduğu güncel veri türünü IDLE ekranında aşağıdaki gibi gösterir:

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

Kod Gözden Geçirme (Code Review) Çıkarımı: Python'un veri türünü anlık olarak değiştirmesi, büyük bir esneklik sağlasa da aynı değişkeni (örneğin asistan_verisi) akış boyunca farklı amaçlarla kullanmak veri tutarsızlığına yol açabilir. Bunun yerine her veri için amacına uygun yeni isimler (asistan_adi, asistan_puani vb.) kullanılmalıdır.

2.1.8. Değişken Tanımlama Kuralları

Programlama dillerinde değişkenler, bellekte tutulan verilere program içerisinden erişmek için kullanılan mantıksal isimlerdir. Python gibi yüksek seviyeli programlama dillerinde programcılar bellek adresleriyle doğrudan ilgilenmek yerine verilere anlamlı değişken adları vererek kodlarını daha okunabilir ve yönetilebilir hâle getirir. Yazılan kaynak kodların Python yorumlayıcısı tarafından doğru şekilde işlenebilmesi için değişken adları belirlenirken Python'un **söz dizimi (syntax)** kurallarına uyulması zorunludur. Ayrıca, okunabilir ve sürdürülebilir kod geliştirmek amacıyla Python'un resmî stil kılavuzu olan PEP 8'de yer alan isimlendirme önerilerine uyulması sektörde yaygın olarak tavsiye edilir. Temel değişken tanımlama kuralları şunlardır:

Karakter Kısıtlaması: Değişken isimleri sadece alfabetik karakterler (A-Z, a-z), rakamlar (0-9) ve alt çizgi (_) karakterini barındırabilir. Bunların dışındaki hiçbir özel sembol (nokta, virgül, soru işareti, tire, boşluk vb.) değişken isminde yer alamaz.

Başlangıç Kriteri: Bir değişken ismi kesinlikle bir rakam ile başlayamaz. İsimler ya alfabetik bir karakterle ya da alt çizgi (_) işaretiyle başlamak zorundadır (Örneğin 3etap_skor hatalı bir söz dizimi iken, etap_skor_3 veya _etap3 kurallara uygundur.).

Büyük-Küçük Harf Duyarlılığı (Case-Sensitivity): Python dili, büyük ve küçük harfleri farklı karakterler olarak değerlendirir. Bu nedenle aynı harflerden oluşsa bile yazımında büyük-küçük harf farkı bulunan kelimeler, bellekte tutulan verilere erişmek için kullanılan farklı değişken adları olarak kabul edilir. Örneğin asistan_adi, Asistan_Adi ve ASISTAN_ADI ifadeleri Python yorumlayıcısı tarafından birbirinden bağımsız üç farklı değişken adı olarak değerlendirilir.

Resmî Dil Anahtarları (Reserved Words): Python dilinin kendi iç mekanizmasını yönetmek ve komutları işletmek adına kendine ayırdığı bazı özel kelimeler (keywords) değişken adı olarak kullanılamaz (Örneğin if, else, True, False, while, for gibi kelimeler Python için özel bir anlam ifade ettiğinden doğrudan değişken ismi yapılamaz.).

Türkçe Karakter Standardı: Python yorumlayıcısı Unicode karakter setini desteklediği için Türkçe karakterleri (ç, ğ, ı, ö, ş, ü) teknik olarak kabul etse de küresel yazılım standartları ve kodun farklı işletim sistemlerinde kararlı çalışabilmesi adına değişken isimlerinde sadece İngiliz alfabesindeki karakterlerin kullanılması önerilir.

2.1.9. Değişken Adlandırma Standartları (Notasyonlar)

Yazılım projelerinde birden fazla kelimeden oluşan değişken isimlerinin okunabilirliğini artırmak amacıyla tüm dünyada kabul görmüş iki temel adlandırma standardı kullanılır.

Yılan Gösterimi (snake_case): Tüm harflerin küçük yazıldığı ve kelimelerin birbirine alt çizgi (_) karakteriyle bağlandığı yöntemdir. Python literatüründe değişken ve fonksiyon isimleri için bu standardın kullanılması (örneğin kullanici_giris_tarihi, asistan_bellek_kapasitesi vb.) kurumsal olarak tavsiye edilir.

Deve Gösterimi (camelCase): İlk kelimenin tüm harflerinin küçük, sonraki her kelimenin ise ilk harfinin büyük yazıldığı ve kelimelerin bitişik (örneğin kullaniciGirisTarihi, asistanBellekKapasitesi) kurgulandığı yöntemdir. Değişken adlandırma standartlarına ilişkin doğru ve hatalı kullanım senaryoları Tablo 2.5'te özetlenmiştir.



Tablo 2.5: Doğru ve Hatalı Değişken Kullanımları Kılavuzu

Değişken Tanımlaması	Durum	Karşılaşılan Söz Dizimi Hatası / Gerekçe
asistan_sürümü = 0.2	Geçerli ancak önerilmez	Teknik olarak çalışsa da küresel standartlar gereği 'ü' harfi yerine 'u' kullanılmalıdır (asistan_surumu).
1.kullanici_adi = "Ali"	Hatalı	SyntaxError: Değişken isimleri rakamla başlayamaz.
kullanici adi = "Veli"	Hatalı	SyntaxError: Kelimeler arasında boşluk karakteri kullanılamaz.
if = "Kod-Asistan"	Hatalı	if ayrılmış anahtar kelime (reserved word) olduğundan veri atanamaz.
kod_asistan_v02 = True	Doğru	Karakter kısıtlamalarına, başlangıç kriterine ve standart notasyona tamamen uygundur.

4. UYGULAMA

Uygulamanın Adı: KOD GÖZDEN GEÇİRME VE SÖZ DİZİMİ DENETİMİ

Uygulamanın Süresi: 20 dakika

Uygulamanın Amacı: Tasarlanan veya aceleyle yazılan kod bloklarındaki değişken adlandırma hatalarını, söz dizimi kurallarına aykırı kullanımları ve isimlendirme problemlerini satır satır incelemek; oluşan hataların nedenlerini belirleyerek uygun çözümler geliştirmek

Problem Senaryosu: Kod-Asistan projesinin kullanıcı profil arayüzünü ve sistem parametrelerini yapılandırmak üzere başka bir yazılım ekibinden ham bir kod taslağı alınmıştır. Kod bloku Python yorumlayıcısına gönderildiğinde ilk satırlardan itibaren **SyntaxError (Söz Dizimi Hatası)** oluştuğu ve programın çalıştırılamadığı görülmüştür. Projenin kararlı ve okunabilir bir yapıya kavuşması için aşağıdaki kaynak kodda bulunan değişken adlandırma ve söz dizimi hatalarının tespit edilerek düzeltilmesi gerekmektedir.

İncelenecek Hatalı Kod Bloku

```
# Sistem Yapılandırma Taslağı
1_asistan_ismi = "KODİ"
asistan_bellek-kapasitesi = "16 GB"
while = 2026
Kullanıcı Adı = "Ebru"
sistem_durumu$ = True
```

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel şekilde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Kodun Kurallara Göre Denetlenmesi

a) Verilen hatalı kod taslağını, değişken tanımlama kriterlerini (karakter kısıtlaması, başlangıç kriteri, resmî dil anahtarları) baz alarak satır satır inceleyiniz.

4. UYGULAMA

b) Tespit ettiğiniz söz dizimi ve değişken adlandırma hatalarını aşağıdaki **Hata Analiz Raporu** alanına gerekçeleriyle birlikte kaydediniz.

Hata Analiz Raporu

1. Satırdaki Akış Hatası ve Gerekçesi: .....
2. Satırdaki Akış Hatası ve Gerekçesi: .....
3. Satırdaki Akış Hatası ve Gerekçesi: .....
4. Satırdaki Akış Hatası ve Gerekçesi: .....
5. Satırdaki Akış Hatası ve Gerekçesi: .....

2. Aşama: Düzeltme Önerisi ve Güvenli Çözüm Üretme

a) Yaptığınız denetim sürecinden hareketle hatalı kod bloklarının Python yorumlayıcısı tarafından kararlı bir şekilde işlenebilmesi için gerekli düzenlemeleri yapınız.

b) Standart adlandırma notasyonlarından snake_case referans alarak düzeltilmiş güvenli kodu yazınız.

Düzeltilmiş Güvenli Kod

```
# Kod-Asistan v0.2 Profil Modülü
.....
.....
.....
.....
.....
```

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme formuna ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

2.1.10. Sabitler

Yazılım geliştirme sürecinde bazı verilerin değerleri programın başlangıcından son adımına kadar sürekli olarak değişebilir, bu tür veriler için değişkenler kullanılır. Bir yazılım mimarisinde programın çalışma süresince değiştirilmemesi amaçlanan ve aynı değeri koruması beklenen verilere **sabit (constant)** adı verilir. Python'da sabitler teknik olarak değiştirilebilir ancak adlandırma kuralları ve geliştirici uzlaşısı gereği değiştirilmemeleri beklenir. Sabitler, yazılımın kararlılığını ve yönetilebilirliğini şu iki temel mekanizma ile güçlendirir:



- **Mantıksal Hataların Önlenmesi:** Büyük harfle yazılan sabit adları, bu değerlerin program akışı içinde değiştirilmemesi gerektiğini geliştiricilere bildirir ve kritik parametrelerin yanlışlıkla değiştirilmesi riskini azaltır.
- **Kod Okunabilirliği ve Kolay Bakım (Maintainability):** Kod bloku içerisinde birden fazla yerde kullanılan ham bir verinin (örneğin yerel bir sunucu adresinin) tek bir merkezden yönetilmesini sağlar. İlgili veri değiştiğinde binlerce satırı düzenlemek yerine sadece en baştaki sabiti güncellemek yeterlidir.

Birçok geleneksel programlama dilinde (C++, Java, C# vb.) bir veriyi sabit yapmak için **const** veya **final** gibi katı söz dizimi anahtarları kullanılır ve bu verinin değeri sonradan değiştirilmek istendiğinde derleyici doğrudan hata verir. Python, yapısı gereği katı bir const mekanizmasına sahip değildir. Python yorumlayıcısı, teknik olarak tüm isimlendirmeleri birer değişken gibi işler. Bu durum, Python dünyasında okunabilirliğini artırmak ve tutarlılığını sağlamak amacıyla Python topluluğunda benimsenen adlandırma geleneğinin (**PEP 8 isimlendirme kuralının**) doğmasını zorunlu kılmıştır. Python'da sabit olarak kullanılması amaçlanan isimler, PEP 8 adlandırma önerisine göre genellikle tüm harfleri büyük olacak ve kelimeler alt çizgiyle (_) ayrılacak biçimde yazılır.

```
# Doğru Sabit Kullanım Standardı (BÜYÜK HARF)
PI_SAYISI = 3.14159
MAKSIMUM_DENEME_SAYISI = 3
SISTEM_PORT_NUMARASI = 443
VARSAYILAN_DIL = "TR"
KARSILAMA_MESAJI = " Sisteme Hoş Geldiniz!"
GELISTIRICI_MODU_ACIK = True
```



Bilgi Kutusu

KODLARIN SESSİZ REHBERİ

Yazılım projelerinde, bilgisayarın görmezden geldiği ancak yazılımcıların birbiriyle iletişim kurmak için bıraktığı "sessiz notlar" bulunur. Python'da # (diyez) işareti ile başlayan bu satırlar, makineye değil doğrudan insana hitap eder. Profesyonel yazılımcılar; kodun ne işe yaradığını takım arkadaşlarına anlatmak veya daha sonra hatırlamak için bu yöntemi kullanır. Siz de kodlarınıza # ile açıklamalar ekleyerek bu alışkanlığı edinebilirsiniz.

Örnek Kullanım

```
# Aşağıdaki değişken kullanıcının hatalı giriş sayısını tutar.
hatali_giris_sayici = 0  # Bu değer her yanlış şifrede 1 artırılmalıdır.
MAKSIMUM_DENEME_SAYISI = 3  # Sabit değer: Sistemin izin verdiği sınır aşılmamalıdır.
# TODO: Gelecek sürümde (v2.0) buraya yeni bir şifre sıfırlama modülü eklenecek.
```

2. ÖRNEK

SEKTÖREL SABİT STANDARDI VE MANTIKSAL DENETİM

Kod-Asistan projesinin kullanıcı giriş panelinde, siber güvenlik önlemi olarak bir kullanıcının en fazla 3 kez hatalı şifre girme hakkı bulunmaktadır. Projenin kararlılığını korumak adına bu üst sınır değeri bir sabit olarak kurgulanmıştır.

Örnek Kod Bloku

```
# Geliştirici Sözleşmesine Uygun Doğru Yapılandırma
MAKSIMUM_DENEME = 3
hatali_giris_sayici = 0  # Bu bir değişkendir, her hatalı girişle değeri artar.
MAKSIMUM_DENEME = 99
```

Kod Analizi

- **Söz Dizimi Durumu**: Başarılı. Python yorumlayıcısı, MAKSIMUM_DENEME = 99 satırında teknik olarak bir çökme veya SyntaxError üretmez, kodu okumaya devam eder.
- **Mantıksal Durum**: Hatalı. Python yorumlayıcısı büyük harfle yazılan bu adı teknik olarak normal bir değişken gibi işler. Ancak PEP 8'e göre bu isim sabit olarak kullanılmak üzere tanımlandığından, değerinin program akışı içinde değiştirilmesi kod incelemesinde mantıksal ve stil açısından sorun olarak değerlendirilir.
- **İnsan Denetimi Kararı**: Kod incelemesinde bu yeniden atama düzeltilmeden kodun ana sürüme alınmaması önerilir.



2. SIRA SİZDE

Aşağıdaki kutucuklarda verilen kavramları anlamlarına uygun olacak şekilde ilgili boşluklara yerleştiriniz.

int	snake_case	bool
float	BÜYÜK_HARF	PEP 8
str		

Programlamada veriler **değişkenler** ve **sabitler** olarak ikiye ayrılır.

- **Değişkenler** isimlendirilirken ..... notasyonu kullanılır.
- **Sabitler** isimlendirilirken ..... notasyonu kullanılır.
- Her ikisinin isimlendirme kurallarının evrensel standardına ..... denir.

Değişkenlerin ve sabitlerin temsil ettiği **Temel Veri Türleri** dörde ayrılır:

- Tam sayılar -> .....
- Ondalıklı sayılar -> .....
- Metinsel ifadeler -> .....
- Mantıksal değerler -> .....

2.2. KULLANICI ETKİLEŞİMİ VE TÜR DÖNÜŞÜMÜ

Algoritma tasarımlarında ve başlangıç seviyesindeki kaynak kod yapılandırmalarında değişkenlerin veya sabitlerin değerleri genellikle doğrudan kaynak kod içinde atanır. Profesyonel yazılım projelerinde ve son kullanıcıya hitap eden sistemlerde veriler programın her çalıştırılma döngüsünde dış dünyadan (kullanıcı klavyesinden, sensörlerden, veri tabanlarından, günlük dosyasından vb.) dinamik olarak sisteme dâhil edilir.

2.2.1. input() ve print() Fonksiyonları

Python dilinde program akışı esnasında kullanıcıdan veri almak ve bu veriyi geçici belleğe aktarmak amacıyla **input()** fonksiyonu kullanılır. Bu yerleşik fonksiyonun arka planda yürüttüğü çalışma mantığı sırasıyla şu üç aşamada gerçekleşir.

- **Akışın Durdurulması:** Python yorumlayıcısı input() komutunun bulunduğu satıra geldiğinde, programın yukarıdan aşağıya doğru işleyen akışını geçici olarak beklemeye alır.
- **Girdi Beklentisi:** Kullanıcı klavyeden bir veri yazıp Enter tuşuna basana kadar sistem beklemeye devam eder.

- **Atama Süreci:** Enter tuşuna basıldığı anda klavyeden girilen tüm karakterler standart girdiden bir metin dizisi olarak döndürülür ve eşitliğin sol tarafındaki değişkene bir veri olarak atanır ve program alt satırdan akmaya devam eder.

Örnek Kod Bloku

```
# Kullanıcıdan veri alma ve bir değişkene atama şeması
kullanici_tahmini = input("Lütfen şifre tahmininizi giriniz: ")
```

Bir programın girdileri alıp işlemesi kadar oluşan sonuçları kullanıcıya anlaşılır, düzenli ve standartlara uygun bir biçimde sunması da etkileşim döngüsünün temel bir parçasıdır. Python programlama dilinde bilgisayar belleğindeki verileri, değişkenleri veya doğrudan metinsel ifadeleri terminal ekranına birer **çıktı** olarak yazdırmak amacıyla yerleşik **print()** fonksiyonu kullanılır.

Örnek Kod Bloku

```
# En temel çıktı üretme biçimi
print("Kod-Asistan sistemine hoş geldiniz!")
```

Yazılım geliştirme süreçlerinde çoğunlukla sadece sabit metinler verilmemelidir, değişken veriler ile metinlerin bir arada sunulması gerekir. Python, bu birleştirme ve ekran çıktısını biçimlendirme işlemi için yazılımcılara kurumsal standartlarda farklı esneklikler sunar.

Virgül (,) ile Bağımsız Argümanları Ayırarak Çıktı Üretme: Farklı veri türündeki değişkenleri ve metinleri yan yana yazdırmak için en güvenli yöntem aralarına virgül koymaktır. Virgül, print() fonksiyonuna verilen bağımsız argümanları ayırır, ekrana yazdırma esnasında verilerin arasına **varsayılan olarak bir adet boşluk** karakteri yerleştirir ve değişkenin türünü değiştirmeden değerin metinsel gösterimini ekrana yazdığı için güvenlidir.

Artı (+) Operatörü ile String Birleştirme (Concatenation): Sadece metinsel ifadeleri birbirine yapışık şekilde bağlamak için kullanılır. Bu operatör, virgülün aksine araya otomatik boşluk bırakmaz. En kritik kararsızlık tuzağı, bir string ile sayısal bir verinin + operatörü ile birleştirilmeye çalışılmasıdır. Bu durum, programın TypeError istisnası üretmesine ve hata yakalanmamışsa yürütmenin durmasına yol açar.

f-String (Formatlı String) Teknolojisi: Modern Python mimarisinde metinlerin içerisine değişkenleri biçimlendirme kolaylığı ve en yüksek okunabilirlikle yerleştirme yöntemidir. String ifadesinin hemen başına f veya F harfi konulur, metin içerisindeki değişkenler ise süslü parantez ({ }) içinde doğrudan çağrılır.

print() fonksiyonu, kaynak kodda açıkça yazılmasa bile arka planda varsayılan değerlerle çalışan, isteğe bağlı iki kritik parametreye sahiptir. Kod yapısının çıktı standartlarını özelleştirmek için bu parametreler manuel olarak manipüle edilebilir.

sep [Separator (Ayırıcı)]: Virgülle ayrılmış ifadelerin arasına hangi karakterin geleceğini belirler. Varsayılan değeri boşluktur (sep=" ").

end [End (Bitiriş)]: Çıktı ekrana basıldıktan sonra satırın nasıl sonlanacağını belirler. Varsayılan değeri bir alt satıra geçme komutu olan yeni satır karakteridir (end="\n").



3. ÖRNEK

DİNAMİK ÇIKTI ÜRETME VE FORMATLAMA (print FONKSİYONU)

Kod-Asistan projesinde asistanın adı, versiyon numarası, sistemin durumu ve kullanıcı logları (kayıtları) terminal ekranına yazdırılmak istenmektedir. Bu süreçte farklı veri tiplerinin birleştirilmesi, metinlerin arasına özel karakterler konulması, bazı çıktıların alt satıra geçmeden aynı satırda devam etmesi gibi çeşitli formatlama ihtiyaçları test edilecektir.

Örnek Kod Bloku

```
# Senaryo A: Virgül (,) ve Artı (+) Operatörlerinin Karşılaştırması
asistan_adi = "Kod Asistanı"
versiyon = 2
# Virgül: Otomatik tür dönüşümü ve otomatik boşluk bırakır.
print("Benim adım", asistan_adi, "Sistem versiyonum:", versiyon)
# Artı (+): Sadece stringleri yapışık bağlar.
karsilama = "Merhaba " + "Yazılımcı"
print(karsilama)
# print("Yaşınız: " + 18) -> TypeError fırlatacağı için bilerek yoruma alındı.
# Senaryo B: F-String (Formatlı String) Teknolojisi
kullanici = "Mustafa"
islem_adedi = 5
print(f"Hoş geldiniz {kullanici}. Bugün yapay zekâ ile {islem_adedi} kod denetlediniz.")
# Senaryo C: print() Fonksiyonunun Gizli Parametreleri (sep ve end)
# sep (Separator): Kelimeler arasına varsayılan boşluk yerine "-" koyar.
print("Siber", "Güvenlik", "Mühendisliği", sep="-")
# end (End): Varsayılan alt satıra geçme (\n) komutunu ezer, aynı satırda tutar.
print("Kod denetimi başladı...", end=" [BAŞARILI] ")
print("Sistem kararlı.")
```

Kod Analizi

- Python'da çıktı üretirken + operatörü tür uyuşmazlığı riski taşıdığı için tehlikelidir.
- Çok sayıda değişken metin içine yerleştirirken en az bellek maliyeti ve en yüksek okunabilirlik sağlayan f-String mimarisi endüstri standardı olarak benimsenmelidir.
- Özel sistem kayıtları üretirken print() fonksiyonunun sep ve end parametrelerine müdahale etmek, geliştiriciye sistem üzerinde tam bir formatlama kontrolü sağlar.

Bilgi Kutusu

Geliştirici Sözleşmesi ve Düzenin Gücü

Python dili, değişken isimlerinin arasına tire (-) koymanıza izin vermez veya sabitlerin küçük harfle yazılmasını teknik olarak engellemez. Küresel yazılım dünyasında milyonlarca yazılımcı, kurallara uymak zorunda olmadıkları hâlde PEP 8 standartlarına ve yılan gösterimi kurallarına harfiyen uyar. Kodları dijital depolara yüklerken her sürüme dürüst ve anlaşılır notlar eklerler. Sizce yazılımcıların kendi aralarında kurdukları bu **Yazılı Olmayan Geliştirici Sözleşmesi** ve iş süreçlerindeki titizlik, dijital dünyadaki karmaşayı nasıl engelliyor olabilir?

5. UYGULAMA

Uygulamanın Adı: İNTERAKTİF VERİ GİRİŞİ VE TÜR DÖNÜŞÜMÜ

Uygulamanın Süresi: 20 dakika

Uygulamanın Amacı: Kod editöründe input() fonksiyonu ile kullanıcıdan veri almak, karşılaşılan veri türü hatalarını bulup çözmek ve int() fonksiyonu ile verileri sayılara dönüştürerek matematiksel hesaplamalar yapmak

Problem Senaryosu: Kod-Asistan projesinin kullanıcı profil modülüne dinamik bir hesaplama fonksiyonu eklenmesi planlanmaktadır. Bu fonksiyon kapsamında, asistanın kullanıcıya doğum yılını sorması ve aldığı bu girdi verisini kullanarak kullanıcının yaşını hesaplayıp ekrana yansıtması istenmektedir. Kodların test aşamasında, kullanıcının girdiği sayısal değerlerin matematiksel işlem sürecine dâhil edilemediği ve sistemin hata vererek durduğu gözlemlenmiştir. Yazılımın kararlı bir şekilde çalışabilmesi için bu veri türü pürüzünün çözülmesi gerekmektedir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Kod Dosyasının Hazırlanması ve Girdi Mekanizması

a) Kod editörü açınız ve sol paneldeki kod_asistanim çalışma klasörü içerisine yas_hesaplayici.py adında yeni bir Python kaynak kod dosyası ekleyiniz.

b) Kullanıcıdan doğum yılı bilgisini dinamik olarak talep etmek amacıyla editör ekranına aşağıdaki komut satırını kodlayınız:

```
dogum_yili = input("Lütfen doğum yılınızı giriniz: ")
```

5. UYGULAMA

2. Aşama: Çalışma Zamanı Hatasıyla Karşılaşma (Metinsel Veri Tuzağı)

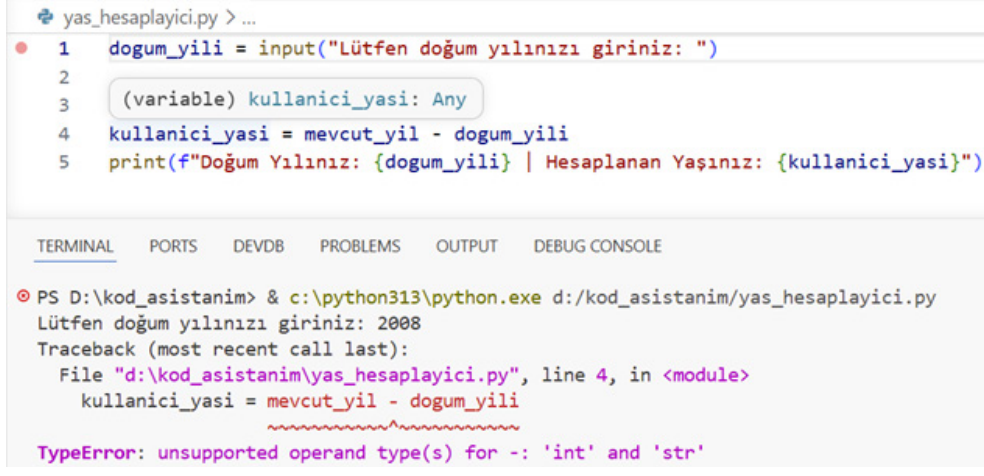
a) Alınan doğum yılı verisini kullanarak yaş hesaplaması yapmak ve sonucu ekrana yazdırmak amacıyla mevcut koda aşağıdaki satırları sırasıyla ekleyiniz:

```
mevcut_yil = 2026
kullanici_yasi = mevcut_yil - dogum_yili
print(f"Doğum Yılınız: {dogum_yili} | Hesaplanan Yaşınız: {kullanici_yasi}")
```

b) Yazılan kod dosyasını çalıştırmak amacıyla editörün sağ üst köşesinde bulunan (Python Dosyasını Çalıştır) butonuna tıklayınız.

c) Terminal panelinde doğum yılı istendiğinde klavyeden herhangi bir sayısal değer (örneğin 2008) yazıp **Enter** tuşuna basınız.

ç) Komut çalıştırıldıktan sonra programın yaş sonucunu üretmediğini ve terminal ekranında bir TypeError: unsupported operand type(s) for -: 'int' and 'str' hatası vererek durduğunu gözlemleyiniz (Görsel 2.7).



Görsel 2.7: Hata ekranı

Hata Analiz Raporu

Python yorumlayıcısı, klavyeden girilen 2008 değerini sayısal bir veri olarak değil metinsel bir karakter dizisi olarak belleğe kaydetmiştir. Matematiksel olarak tam sayı türündeki bir veriden (2026), metinsel türdeki bir verinin (2008) çıkarılması imkânsız olduğundan Python yorumlayıcısı, uyumsuz veri türleri nedeniyle TypeError üretmiş ve programın yürütülmesini durdurmuştur.

5. UYGULAMA

3. Aşama: Tip Dönüşümü ile Sorunun Çözülmesi

a) Hatayı gidermek ve input() fonksiyonundan gelen metinsel veriyi matematiksel işlemlere uygun bir tam sayıya dönüştürmek amacıyla kod blokunu aşağıdaki şekilde yeniden düzenleyiniz.

```
# Veri girişi anında int() fonksiyonu ile tip dönüşümü yapılandırılıyor.
dogum_yili = int(input("Lütfen doğum yılınızı giriniz: "))
mevcut_yil = 2026
kullanici_yasi = mevcut_yil - dogum_yili
print(f"Doğum Yılınız: {dogum_yili} | Hesaplanan Yaşınız: {kullanici_yasi}")
```

b) Düzenlenen kodu, sağ üstteki **Run Python File** butonuna tıklayarak yeniden çalıştırınız. Bu kez sistemin sorunsuz çalıştığını ve ekrana kararlı bir şekilde Hesaplanan Yaşınız: 18 çıktısını yansıttığını gözlemleyiniz.

4. Aşama: Geliştirilen Yeni Modülün Bulut Deposuna Eklenmesi

Yazılımcılar, projeye kararlı bir şekilde çalışan her yeni fonksiyon veya kaynak kod dosyası eklediklerinde bu güncel durumu dijital depoya yansıtmakla yükümlüdürler. VS Code terminalinde başarıyla test ettiğiniz yas_hesaplayici.py kod dosyasını GitHub deposuna yüklemek için aşağıdaki basamakları takip ediniz.

a) İnternet tarayıcınızdan GitHub hesabınızı açınız ve daha önce oluşturduğunuz kod-asistan isimli proje deponuza giriş yapınız.

b) Depo arayüzünün sağ üst tarafında bulunan **Add file (Dosya ekle)** butonuna tıklayarak **Upload files (Dosyaları yükle)** seçeneğini işaretleyiniz.

c) Bilgisayarınızın yerel proje klasöründeki kararlı çalışan güncel yas_hesaplayici.py kaynak kod dosyasını sürükleyip yükleme alanına bırakınız.

ç) Sayfa altında yer alan sürüm notu (Commit text) alanına, yapılan işi net ve sistematik bir dille açıklayan şu kurumsal notu ekleyiniz: Profil modülüne yaş hesaplama fonksiyonu entegre edildi.

d) **Commit changes** butonuna tıklayarak kodlarınızı dijital deponuza yükleyiniz.

Yazılım projelerinde sadece tam sayılarla (int) çalışmak, finansal işlemler veya hassas ölçümler gerektiren durumlarda yetersiz kalır. Döviz kurları, ürün fiyatları, ağırlık gibi ondalıklı değer barındıran kullanıcı girdilerinin matematiksel hesaplamalara dâhil edilebilmesi için metinsel (str) verilerin float() fonksiyonu kullanılarak ondalıklı sayı türüne dönüştürülmesi gereklidir. Arka planda hesaplanan bu verilerin kullanıcıya sunulurken bilgilendirici metinlerle birleştirilmesi, yazılımın okunabilirliğini ve kalitesini artırır.

6. UYGULAMA

Uygulamanın Adı: DİNAMİK DÖVİZ VE AKILLI KUMBARA ASİSTANI

Uygulama Süresi: 20 Dakika

Problem Tanımı: Kod-Asistan yazılımına, kullanıcıların birikimlerini takip eden akıllı bir finansal modül eklemeniz istenmektedir. Kullanıcı, her işlemde kumbarasına eklediği Türk Lirası (TL) miktarını ve o günkü güncel döviz kurunu klavyeden sisteme girecektir. Program, kullanıcıdan alınan bu verileri işleyerek kumbaraya eklenen TL tutarını ve bunun döviz karşılığını hesaplamalı; sonucu f-string kullanarak ekrana yazdırmalıdır. Kullanıcı döviz kurunu ondalıklı bir değer (örneğin 34.55) olarak girebileceğinden, giriş verilerinin uygun veri tipine dönüştürülmesi ve hesaplamalarda doğru veri türleriyle kullanılması gerekmektedir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel şekilde gerçekleştiriniz.

Uygulama Adımları

a) Kod editörünü açınız ve kod_asistanim çalışma klasörünüzün içerisine kumbara_asistani.py adında yeni bir temiz Python kaynak kod dosyası ekleyiniz.

b) Kullanıcıdan kumbaraya atılan TL miktarını ve bunun güncel Döviz kurunu input() fonksiyonu ile dinamik olarak talep ediniz.

c) Alınan verilerin varsayılan olarak metinsel (str) olduğunu unutmayınız, matematiksel bölme işlemine dâhil edebilmek için kur ve TL verilerini uygun sayısal veri türlerine (float) dönüştürünüz.

ç) Döviz karşılığını hesaplayan matematiksel formülü kurunuz ve oluşan sonucu f-string mimarisiyle ekrana yazdırınız.

d) Değişken isimlerinizde snake_case notasyonunu kusursuzca uygulayınız.

e) Kodunuzu kod editörü terminalinde test ettikten sonra dijital deponuza giderek "Dinamik kumbara asistanı modülü tip dönüşümleriyle kararlı şekilde kodlandı." sürüm notuyla (commit) dijital depoya yükleyiniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme formuna ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

2.2.2. Temel Tür Dönüşüm Fonksiyonları ve Sınır Durum Analizi

Yazılım geliştirme süreçlerinde dinamik veri girişi sağlayan input() fonksiyonunun girdileri, her zaman metinsel veri türünde belleğe kaydedilir. Girdilerin sayısal algoritmalara dâhil edilebilmesi için kullanılan int() fonksiyonu gibi Python dilinde her temel veri türünün kendi adıyla anılan, yerleşik bir tip dönüşüm fonksiyonu bulunur.

Verilerin bir türden diğerine dönüştürülmesi; bilgisayar belleğindeki orijinal verinin değiştirilmesi değil, o veriden istenen yeni türde tamamen yeni bir değer (nesne) üretilmesi ve veri üzerinde yürütülebilecek mantıksal veya aritmetik işlem yeteneklerinin yeniden tanımlanması anlamına gelir.

float() Fonksiyonu: Kendisine parametre olarak verilen tam sayıları veya sayısal içerikli metinleri ondalıklı sayı türüne dönüştürür. Sayısal değeri kayan noktalı sayı türünde temsil eder.

str() Fonksiyonu: Sayısal veya mantıksal değerleri, metinsel karakter dizilerine dönüştürür. Bu dönüşümden sonra veriler üzerinde matematiksel hesaplamalar yapılamaz hâle gelir.

bool() Fonksiyonu: Verilen değerlerin mantıksal birer doğruluk değeri (True / False) üretmesini sağlar. Python bellek mimarisinde **sıfır (0 ve 0.0)**, **"" (boş metin)**, **boş koleksiyonlar, None** gibi değerler False kabul edilirken, bunların dışındaki çoğu değer True olarak anlamlandırılır.

Yazılımcının kontrolü dışında dönüştürülmesi teknik olarak imkânsız olan veriler fonksiyonlara parametre olarak gönderildiğinde dönüşüm hatası oluşur ve bu hata yakalanmazsa program durur. Bu sınır durumlar ve hata dinamikleri şu şekildedir:

Metinden Sayıya Dönüşüm Sınırı: İçerisinde alfabetik karakter veya özel sembol barındıran bir metin, int() veya float() fonksiyonu ile sayıya dönüştürülemez [Örneğin int("Kod Asistan") ifadesi doğrudan ValueError hatasını geri gönderir.].

Ondalıklı Metin Tuzağı: Doğrudan tam sayıya çevrilemeyen ondalıklı bir metin ifadesi önce float() fonksiyonuyla sayıya dönüştürülmeli, ardından int() katmanına alınmalıdır [Örneğin int("99.9") satırı hata verirken int(float("99.9")) satırı kararlı çalışarak 99 çıktısını üretir.]. Burada int() fonksiyonunun matematiksel bir yuvarlama yapmadığına, sadece sayının ondalık kısmını kesip attığına dikkat ediniz

bool() Fonksiyonunda Yokluk Denetimi: Sayısal olarak 0 ve 0.0 değerleri ve metinsel olarak tamamen boş bırakılmış "" ifadeleri bool() süzgecinden geçirildiğinde False sonucunu üretir. İçinde tek bir boşluk karakteri dahi olan " " ifadesi ise bellek hücresi dolu kabul edildiği için True çıktısını basar.



Tablo 2.6, farklı kaynak verilerin ilgili dönüşüm fonksiyonlarına tabi tutulduklarında ürettikleri sonuçları ve bellek çıktı analizi durumlarını göstermektedir.

Tablo 2.6: Veri Türü Dönüşüm Çıktıları ve Sınır Durumları

Kaynak Veri	Kullanılan Fonksiyon	Üretilen Nihai Çıktı	Elde Edilen Veri Türü	Sistem Kararlılık Durumu
yas = "18"	int(yas)	18	int	Kararlı: Matematiksel hesaplamalara uygundur.
skor = "A5"	int(skor)	HATA	—	ValueError: Metin içinde rakam dışı karakter vardır.
fiyat = 89	float(fiyat)	89.0	float	Kararlı: Değer float türüne dönüştürüldü.
kod = "3.14"	int(kod)	HATA	—	ValueError: Ondalıklı metin doğrudan int olamaz.
veri = 100	str(veri)	"100"	str	Kararlı: Karakter dizisi oldu, aritmetiğe kapatıldı.
durum = 0	bool(durum)	False	bool	Kararlı: Sıfır, Python'da yanlış kabul edilen değerlerden biridir.
metin = ""	bool(metin)	False	bool	Kararlı: Boş karakter dizisi yokluk ifade eder.
bosluk = " "	bool(bosluk)	True	bool	Kararlı: Hücrede boşluk karakteri barındığı için doludur.



Düşünelim-Tartışalım

Bir e-ticaret sisteminin fatura modülü tasarlanırken kullanıcının klavyeden girdiği **fatura_tutari = "250.75"** değeri önce float() ile dönüştürülüp ardından int() fonksiyonuna verilirse hangi veri kaybı yaşanır? Verinin bellek davranışı boyutunu dikkate alarak açıklayınız.

2.3. OPERATÖRLER VE İŞLEM ÖNCELİĞİ

Yazılım geliştirme süreçlerinde verilerin sadece belleğe kaydedilmesi veya türlerinin dönüştürülmesi tek başına bir anlam ifade etmez. Programların işlevsel birer ürüne dönüşebilmesi için bu verilerin belirli kurallar çerçevesinde işlenmesi, birbiriyle kıyaslanması ve mantıksal karar mekanizmalarına dâhil edilmesi gerekir.

2.3.1. Operatörler

Bilgisayarlarda veriler üzerinde matematiksel, karşılaştırmalı veya mantıksal yönlendirmeler yapmasını sağlayan özel sembollere ve karakter dizilerine **operatör** adı verilir. Operatörlerin işleme aldığı değişken ya da sabit değerlere ise **operand** denir.

Aritmetik operatörler; matematiksel hesaplamaların, formüllerin ve sayısal veri analizlerinin kaynak kod içerisinde çalıştırılmasını sağlayan sembollerdir. Python dilinde kullanılan standart aritmetik operatörlerin işlevleri ve bellek çıktı davranışları şöyle yapılandırılmıştır:

Toplama (+): İki sayısal veriyi toplar. Metinsel veriler arasında kullanıldığında ise birleştirme (concatenation) işlevi görür.

Çıkarma (-): Soldaki sayısal değerden sağdaki sayısal değeri çıkarır.

Çarpma (*): İki sayısal değeri çarpar. Metinsel bir veri ile bir tam sayı arasında kullanıldığında metni belirtilen sayı kadar ardışık olarak çoğaltır.

Bölme (/): Soldaki değeri sağdaki değere böler. Python bellek mimarisinde bu operatörün ürettiği sonuç, bölünen sayılar tam sayı olsa dahi her zaman ondalıklı sayı (float) türünde tahsis edilir.

Taban Bölme / Tam Bölme (//): Bölme sonucunun taban değerini, yani kendisinden büyük olmayan en yakın tam sayıyı üretir.

Mod Alma (%): Bölme işlemi sonucunda kalan sayısal değeri sisteme döndürür. Özellikle bir sayının tek mi, çift mi olduğunun denetlenmesinde (2'ye göre mod alma) sıklıkla kullanılır.

Üs Alma (**): Soldaki sayının, sağdaki sayı kadar kuvvetini (üssünü) hesaplar. (Örneğin 2**3 ifadesi karşılığı olarak 8 çıktısını üretir.).

Python programlama dilinde sık kullanılan bu aritmetik operatörlerin işlevleri, örnek çıktı sonuçları ve bellek durumları Tablo 2.7'de detaylı olarak özetlenmiştir.

Tablo 2.7: Python Aritmetik Operatörleri ve Çıktı Analizleri

İşlem Kelimesi / İfadesi	Kullanılan Operatör	Çıktı Sonucu	Elde Edilen Veri Türü	Teknik Açıklama
sonuc = 10 / 4	/	2.5	float	Bölme işleminin sonucu her zaman ondalıklı (float) veri türünde üretilir.
sonuc = 10 // 4	//	2	int	Tam bölme işlemi, sonucun ondalıklı kısmını kaldırarak tam sayı değerini döndürür.
sonuc = 13 % 5	%	3	int	Mod alma işlemi, bölme sonucunda kalan değeri üretir.
sonuc = 5 ** 2	**	25	int	Üs alma işlemi, sayının belirtilen kuvvetini hesaplayarak sonucu üretir.

Karşılaştırma operatörleri; iki verinin birbiriyle olan büyüklük, küçüklük veya eşitlik durumlarını analiz etmek amacıyla kullanılan matematiksel denetim araçlarıdır. Karşılaştırma (İlişkisel) operatörleri, uyumlu veri türleri arasında kullanıldığında sisteme her zaman mantıksal veri türünde sadece **True** veya **False** çıktısı üretir.

- **Eşittir (==):** İki tarafın değerlerinin birbirine eşit olup olmadığını denetler. Eşit ise True, değil ise False üretir. (Tek eşittir = sembolü bellekte veri atama eylemidir, çift eşittir == ise mantıksal kıyaslama operatörüdür.).
- **Eşit Değildir (!=):** İki tarafın değerlerinin birbirinden farklı olup olmadığını denetler. Değerler birbirinden farklı ise True, aynı ise False sonucunu verir.
- **Büyüktür (>) ve Küçüktür (<):** Soldaki değerin sağdakinden büyüklük veya küçüklük durumunu denetler.



- **Büyük Eşittir (>=) ve Küçük Eşittir (<=):** Değerlerin hem büyüklük / küçüklük hem de eşitlik sınır durumlarını aynı anda kontrol eder.

Mantıksal operatörler; birden fazla karşılaştırma ifadesinin veya mantıksal durumun birleştirilerek tek bir nihai karar mekanizmasına dönüştürülmesini sağlayan operatörlerdir. Özellikle program akışındaki çoklu şart senaryolarının denetlenmesinde kritik rol oynarlar.

- **ve (and) Operatörü:** Bağladığı tüm karşılaştırma ifadelerinin veya mantıksal durumların **tamamının** True olması durumunda True sonucunu üretir. Koşullardan tek bir tanesi bile False olursa nihai çıktı doğrudan False olur.
- **veya (or) Operatörü:** Bağladığı koşullardan **en az bir tanesinin** True olması durumunda True çıktısını basar. Sistemin False üretmesi için tüm koşulların aynı anda False olması gerekir.
- **değil (not) Operatörü:** Önüne geldiği mantıksal değerin veya karşılaştırma sonucunun tersini alır. Bu operatör; True olan bir durumu False hâline getirir, False olan bir durumu ise True olarak tersine çevirir.

Python programlama dilinde sık kullanılan operatörlerin işlevleri, örnek çıktı sonuçları Tablo 2.8'de detaylı olarak özetlenmiştir.

Tablo 2.8: Python Operatörleri ve Çıktı Analizleri

İşlem Kelimesi / İfadesi	Kullanılan Operatör	Çıktı Sonucu	Elde Edilen Veri Türü	Teknik Açıklama
sonuc = 10 / 4	/	2.5	float	Bölme işlemi otomatik olarak float hücre tahsis eder.
sonuc = 10 // 4	//	2	int	Ondalık kısım (0.5) bellekten atılarak tam sayı bırakılır.
sonuc = 13 % 5	%	3	int	13'ün 5 ile bölümünden kalan sayısal değer üretilir.
kontrol = (5 == 5)	==	True	bool	İki değer birbirine eşit olduğu için koşul sağlanır ve True sonucu üretilir.
kontrol = (10 != 10)	!=	False	bool	Değerler birbirine eşit olduğu için "eşit değildir" koşulu sağlanmaz ve False sonucu üretilir.
onay = (5 > 2) and (3 < 1)	and	False	bool	Koşullardan biri yanlış (False) olduğu için and (ve) kuralı gereği sonuç False olur.
onay = (5 > 2) or (3 < 1)	or	True	bool	İlk koşul doğru (True) olduğu için or (veya) kuralı gereği tek bir şartın sağlanması yeterlidir ve sonuç True olur.
onay = not (10 > 20)	not	True	bool	Parantez içi False sonucunu verir; not (değil) operatörü bu değeri tersine çevirerek True yapar.

Düşünelim-Tartışalım

Bir bilgisayar yazılımı tasarlanırken sisteme girilen bir kullanıcı şifresinin **hem en az 8 karakterden oluşması hem de içinde en az bir adet özel sembol barındırması** kurallarının aynı anda kontrol edilmesi istenmektedir. Bu iki şartın kontrolü için ilişkisel operatörlerin ardından hangi **mantıksal operatör** kullanılmalıdır? Nedenini gerekçeleriyle açıklayınız.

7. UYGULAMA

Uygulamanın Adı: ARİTMETİKSEL VE MANTIKSAL ANALİZ

Uygulamanın Süresi: 20 dakika

Uygulamanın Amacı: Farklı operatör gruplarının (aritmetik, ilişkisel, mantıksal) çalışma mekanizmalarını kaynak kodlar üzerinden test etmek, bölme operatörlerinin (/, //) ürettiği sonuçları karşılaştırmak ve karmaşık mantıksal koşulların ürettiği bool sonuçlarını terminal ekranında analiz etmek

Problem Senaryosu: Kod-Asistan projesinin merkezî karar ve matematiksel işlem modülünün v0.2 sürüm taslağı hazırlanmaktadır. Bu modül kapsamında asistanın, sisteme dâhil edilen sayısal veriler üzerinde taban bölme ve kalan hesaplama işlemlerini yürütmesi, aynı zamanda kullanıcı profilinin doğruluğunu çoklu mantıksal süzgeçlerden geçirerek denetlemesi istenmektedir. Sistem bileşenlerinin kararlı ve doğru çıktılar üretebilmesi için operatörlerin işlem davranışlarının bilgisayar ortamında test edilerek doğrulanması gerekmektedir.

Verilen senaryodan hareketle uygulamanın aşamalarını sırasıyla takip ederek uygulamayı gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Kod Dosyasının Hazırlanması ve Aritmetik Analiz

a) Bilgisayardaki kod editörü programını açınız ve sol paneldeki kod_asistanim çalışma klasörü içine operator_analizi.py adında yeni bir Python kaynak kod dosyasını ekleyiniz.

b) Bölme (/) ile taban bölme (//) operatörleri arasındaki bellek ve veri türü farkını gözlemlemek amacıyla editör ekranına aşağıdaki komut satırlarını kodlayınız:

```
# 1. Aşama: Aritmetik Operatörlerin İncelenmesi
normal_bolme = 10 / 4
taban_bolme = 10 // 4
kalan_bulma = 10 % 4
print("Normal Bölme Sonucu:", normal_bolme)
print("Taban Bölme Sonucu:", taban_bolme)
print("Bölme İşleminden Kalan:", kalan_bulma)
```

7. UYGULAMA

Yazılan kod dosyasını çalıştırmak amacıyla editörün sağ üst köşesinde bulunan **Run Python File** butonuna tıklayınız ve Terminal panelindeki çıktıları inceleyiniz.

2. Aşama: İlişkisel ve Mantıksal Karar Yapılarının Entegrasyonu

a) Analiz sürecine mantıksal operatörlerin etki mekanizmasını dâhil etmek amacıyla mevcut kod dosyasının alt satırlarına sırasıyla aşağıdaki komut blokları ekleyiniz:

```
# 2. Aşama: İlişkisel ve Mantıksal Operatörlerin İncelenmesi
sayi1 = 15
sayi2 = 20
# Çoklu koşul denetimi (and ve or operatörlerinin davranışı)
kosul_ve = (sayi1 > 10) and (sayi2 < 15)
kosul_veya = (sayi1 > 10) or (sayi2 < 15)
kosul_degil = not (sayi1 == sayi2)
print("ve (and) Koşul Çıktısı:", kosul_ve)
print("veya (or) Koşul Çıktısı:", kosul_veya)
print("değil (not) Koşul Çıktısı:", kosul_degil)
```

b) Düzenlenen kod dosyasını, sağ üstteki **Run Python File** butonuna tıklayarak yeniden çalıştırınız ve terminal ekranına yansıyan nihai çıktıları satır satır analiz ediniz.

Terminal Çıktısı Analizi

Kaynak kodlar hatasız şekilde çalıştırıldığında Python yorumlayıcısının operatör sembollerini işleyerek ürettiği kararlı çıktılar terminal paneline şu şekilde yansıtılır:

```
Normal Bölme Sonucu: 2.5
Taban Bölme Sonucu: 2
Bölme İşleminden Kalan: 2
ve (and) Koşul Çıktısı: False
veya (or) Koşul Çıktısı: True
değil (not) Koşul Çıktısı: True
```

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme formuna ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

2.3.2. İşlem Önceliği Kuralları

Kaynak kodlar içerisinde birden fazla operatörün aynı satırda ardışık olarak kullanılması durumunda, bilgisayarın hangi işlemi daha önce yürüteceğini belirleyen kurallar bütününe **işlem önceliği (precedence)** denir. Tıpkı geleneksel matematikteki çarpma işleminin toplama işlemine göre öncelikli olması gibi programlama dillerinde de her operatör grubunun hiyerarşik bir baskınlık sırası mevcuttur.

Bu kuralların yazılımcı tarafından dikkate alınmaması, programın çalışma zamanında çökmesine yol açmasa da üretilen sonuçların yanlış hesaplanmasına bir diğer ifadeyle **mantıksal hatalara (logic errors)** neden olur.

Verilen liste, Python yorumlayıcısının karmaşık kod satırlarını anlamlandırırken yukarıdan aşağıya doğru takip ettiği işlem önceliği sırasını gösterir.

1. **Parantezler ( ):** En yüksek önceliğe sahiptir. Karmaşık aritmetik veya mantıksal ifadelerin öncelik sırasını manuel olarak değiştirmek ve kodu daha okunabilir kılmak amacıyla kullanılır.
2. **Üs Alma **:** Sayısal kuvvet hesaplamaları, tüm aritmetik işlemlerden önce çalıştırılır.
3. **Çarpma, Bölme, Taban Bölme, Mod %, /, //, *:** Bu operatörler kendi aralarında **aynı öncelik düzeyine** sahiptir.
4. **Toplama ve Çıkarma +, -:** Aritmetik operatörler arasında en son çalıştırılan gruptur.
5. **Karşılaştırma Operatörleri ==, !=, >, <, >=, <=:** Aritmetik hesaplamaların tamamlanmasının ardından ilişkisel denetimler devreye girer.
6. **Mantıksal Operatörler not, and, or:** Karar mekanizmalarında en son işlenen gruptur. Kendi içindeki hiyerarşi sırasıyla önce not, sonra and, en son or şeklindedir.

Eğer aynı kod satırında, öncelik derecesi birbiriyle tamamen eşit olan operatörler (örneğin yan yana hem çarpma hem de bölme) yer alıyorsa Python yorumlayıcısı işlemi standart olarak **soldan sağa doğru** ardışık bir hiyerarşiyle yürütür ancak üs alma () operatörü bu kuralın istisnasıdır ve aynı satırda ardışık kullanıldığında sağdan sola doğru değerlendirilir.



4. ÖRNEK

İŞLEM ÖNCELİĞİ ETKİSİ

Kod-Asistan projesinde kullanıcının kilo ve boy verileri alınarak Vücut Kitle İndeksi (VKİ) hesaplanacaktır. VKİ formülü Kilo / (Boy x Boy) şeklindedir. Bilgisayar işlemcisinin karmaşık durumlarda hangi işlemi daha önce yürüteceğini belirleyen hiyerarşik yerleşimlerin ve parantez kullanımının, bilgisayar belleğindeki hesaplama sonuçlarını nasıl doğrudan değiştirdiği bu senaryo üzerinden test edilecektir.

Örnek Kod Bloku

```
# Senaryo A: Parantez Kullanılmayan Hatalı Durum (Mantık Hatası)
kilo = 70.0
boy = 1.75
# Python, aynı öncelikteki çarpma ve bölme işlemlerini soldan sağa değerlendirir. Önce kiloyu boya böler (70/1.75=40), sonra boyla çarpar (40*1.75=70).
hatali_vki = kilo / boy * boy
print("Hatalı Hesaplama:", hatali_vki)
# Terminal Çıktısı: 70.0 (Mantıksız Sonuç)
# Senaryo B: Parantez ile İşlem Önceliğinin Yönetilmesi (Güvenli Çözüm)
# Parantez içi en yüksek önceliğe sahip olduğundan önce boy karesi (boy*boy) hesaplanır, sonra bölme yapılır.
dogru_vki = kilo / (boy * boy)
print(f"Doğru Hesaplama: {dogru_vki:.2f}")
# Terminal Çıktısı: 22.85 (Doğru Sonuç)
```

Kod Analizi

- Karmaşık hesaplama satırlarında Python yorumlayıcısının varsayılan hiyerarşisi dikkate alınmalıdır.
- Hem kodun diğer yazılımcılar tarafından daha rahat okunabilmesi hem de olası mantık hatalarının engellenmesi adına öncelikli çalışması istenen bloklar mutlaka parantez ( ) içerisine alınarak güvence altına alınmalıdır.



Düşünelim-Tartışalım

Bir öğrencinin üç farklı sınav notunun ortalamasını hesaplamak amacıyla kaynak koda **ortalama = not1 + not2 + not3 / 3** satırı yazılmıştır. Bu kod satırı çalıştırıldığında Python yorumlayıcısı matematiksel olarak nasıl bir öncelik hatası yapar? Doğru ortalama sonucuna ulaşmak için kod satırı nasıl revize edilmelidir? Gerekçeleriyle açıklayınız.

2.4. YAPAY ZEKÂ KOD DENETİMİ

Yazılım dünyasında büyük dil modelleri ve yapay zekâ teknolojileri; kaynak kod alternatifleri üretme, algoritma tasarlama ve yazım hatalarını saptama süreçlerinde yazılımcılara rehberlik eden güçlü birer yardımcı hâline gelmiştir. Yapay zekâ modellerinin kod üretebiliyor olması, üretilen bu kodların her zaman kararlı, verimli ve hatasız olduğu anlamına gelmez.

Yapay zekâ araçları, insan düşüncesinin mimari ve mantıksal derinliğine sahip değildir. Büyük dil modelleri, eğitim verileri ve verilen bağlama dayanarak olası kod dizilerini üretir ancak kodun doğruluğunu veya güvenliğini garanti etmez. Bu teknik doğalarından ötürü, bazen gerçekte var olmayan komutları varmış gibi üretebilir ya da ilk bakışta doğru görünen fakat arka planda ciddi mantık hataları barındıran kod blokları ortaya çıkarabilir. Yazılım literatüründe bu duruma **yapay zekâ halüsinasyonu (AI hallucination)** adı verilir.

2.4.1. İnsan Denetimi İlkesi

Yapay zekânın ürettiği kaynak kodların barındırabileceği mantıksal, aritmetiksel kusurların veya söz dizimi kusurlarının yazılımı kararsızlaştırmasını önlemek adına endüstriyel standartlarda **İnsan Denetimi (Human-in-the-Loop)** prensibi uygulanır. İnsan Denetimi prensibi; yapay zekâyı karar verici mutlak bir otorite olarak değil yalnızca bir **üretim asistanı** olarak konumlandırır. Bu yaklaşıma göre yapay zekâ tarafından üretilen her bir kod satırı, algoritma mimarı olan insan tarafından satır satır denetlenmeli ve kod, üretim sistemine alınmadan önce insan tarafından incelenmeli ve güvenli bir test ortamında çalıştırılarak doğrulanmalıdır.

2.4.2. Yapay Zekâ Destekli Yazılım Geliştirmede Sorumluluk Sınırı

Bir yazılım projesinde yapay zekâ yardımı alınsa dahi kodun kararlı çalışmasından, veri türlerinin doğru seçilmesinden ve işlem önceliği kurallarının hatasız işletilmesinden doğrudan **algoritmayı tasarlayan yazılımcı sorumludur**. Yapay zekânın sunduğu önerileri körü körüne koda dâhil etmek, projenin çökmesine veya bellek kayıplarına yol açabilir. Profesyonel bir bilişim uzmanı, asistanın ürettiği ham kodları birer taslak kabul ederek kendi teknik bilgisi ve insan zekâsıyla kodlara nihai şeklini verir.



Düşünelim-Tartışalım

Bir web sitesinin üyelik modülü için yapay zekâya kod ürettirildiğinde yapay zekânın değişken isimlerini kurallara aykırı olarak boşluklu veya rakamla başlayacak şekilde tasarladığını varsayınız. İnsan Denetimi prensibi doğrultusunda bu kod blokunu doğrudan sisteme aktarmak yerine nasıl bir süreç işletilmelidir? Gerekçeleriyle açıklayınız.

8. UYGULAMA

YAPAY ZEKÂ KOD DOĞRULAMA VE TÜR HATASI DENETİMİ

Uygulamanın Süresi: 20 dakika

Uygulamanın Amacı: Yapay zekâ tarafından üretilen ham kaynak kodlarını İnsan Denetimi prensibiyle incelemek, kodun içindeki veri türü ve işlem önceliği hatalarını araştırmacı ve sorgulayıcı bir yaklaşımla saptayarak hatasız algoritmayı yeniden yapılandırmak

Problem Senaryosu: Kod-Asistan projesinin mini matematik modülü için bir yapay zekâ asistanından ham kod taslağı talep edilmiştir. Yapay zekâ, kullanıcıdan iki sayı alıp ortalamasını ekrana döndüren bir program hazırlamıştır. Bu kod bloku, insan süzgecinden geçirilmeden doğrudan sisteme dâhil edilirse programın veri türü uyuşmazlığı nedeniyle hata üreteceği saptanmıştır. Sizden beklenen araştırmacı bir kimlikle bu kod blokundaki tür hatasını sorgulayıp bulmanız, eksik bırakılan işlem önceliği parantezlerini koda yerleştirerek programı kararlı hâle getirmenizdir.

Yapay Zekâ Tarafından Üretilen Ham Taslak (Hatalı Kod)

```
# Kod-Asistan v0.2 - Yapay Zekânın Ürettiği Hatalı Taslak
sayi1 = input("Birinci sayıyı giriniz: ")
sayi2 = input("İkinci sayıyı giriniz: ")
ortalama = sayi1 + sayi2 / 2
print("Girdiğiniz sayıların ortalaması:", ortalama)
```

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulama Adımları

a) Kod editörünüzde yapay_zeka_denetimi.py adında yeni bir Python dosyası oluşturarak hatalı kod taslağını yazıp çalıştırınız.

b) Terminal ekranında sayılar girildiğinde, programın ürettiği hatayı ve matematiksel kusurları sorgulayarak problemin kök nedenini saptayınız.

c) Veri türü dönüşümü ve işlem önceliği eksikliklerini gidererek kodu güvenli hâle getiriniz.

ç) Hatasız hâle getirdiğiniz kararlı kodunuzu, "Yapay zekâ kodundaki tür hatası ve işlem önceliği kusuru İnsan Denetimi prensibiyle giderildi." sürüm notuyla dijital deponuza yükleyiniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme formuna ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

2. SIRA SİZDE

1. Öğrenme birimi boyunca gerçekleştirdiğiniz tüm kodlama etkinliklerinizi, geliştirdiğiniz algoritmaları, veri tipi dönüşümleriyle ilgili çalışmalarınızı ve Kod-Asistan çalışmalarınızı gözünüzün önünden geçiriniz.
2. Aşağıda üç farklı geometrik sembolle tanımlanan alanlardaki (Üçgen, Kare, Çember) yönlendirme sorularını dikkatlice okuyunuz.
3. Öğrenme birimi konularıyla ilgili öğrendiklerinizden hareketle kendi öğrenme yolculuğunuzu yansıtan içten ve özgün cevaplarınızı verilen boşluklara yazınız.

ÜÇGEN: Zihnimde Şekillenen 3 Temel Kazanım

Bu öğrenme biriminin sonunda artık tam anlamıyla uzmanlaştığımı düşündüğüm, başkasına hiç zorlanmadan anlatabileceğim **3 temel algoritmik beceri / bilgi** şunlardır:

1. .....
2. .....
3. .....

KARE: Zihnimi "Köşeye Sıkıştıran" ve Netleşmesi Gereken Pürüzler

Kodlama ortamında kod yazarken, input() fonksiyonundaki veri türü tuzağını yönetirken, karmaşık işlem önceliği hiyerarşilerini kurgularken veya mantıksal operatörleri bağlarken beni hâlâ zorlayan, zihnimde tam olarak oturmayan ve **öğretmenime sormam gereken köşeli / pürüzlü noktalar** şunlardır:

- .....

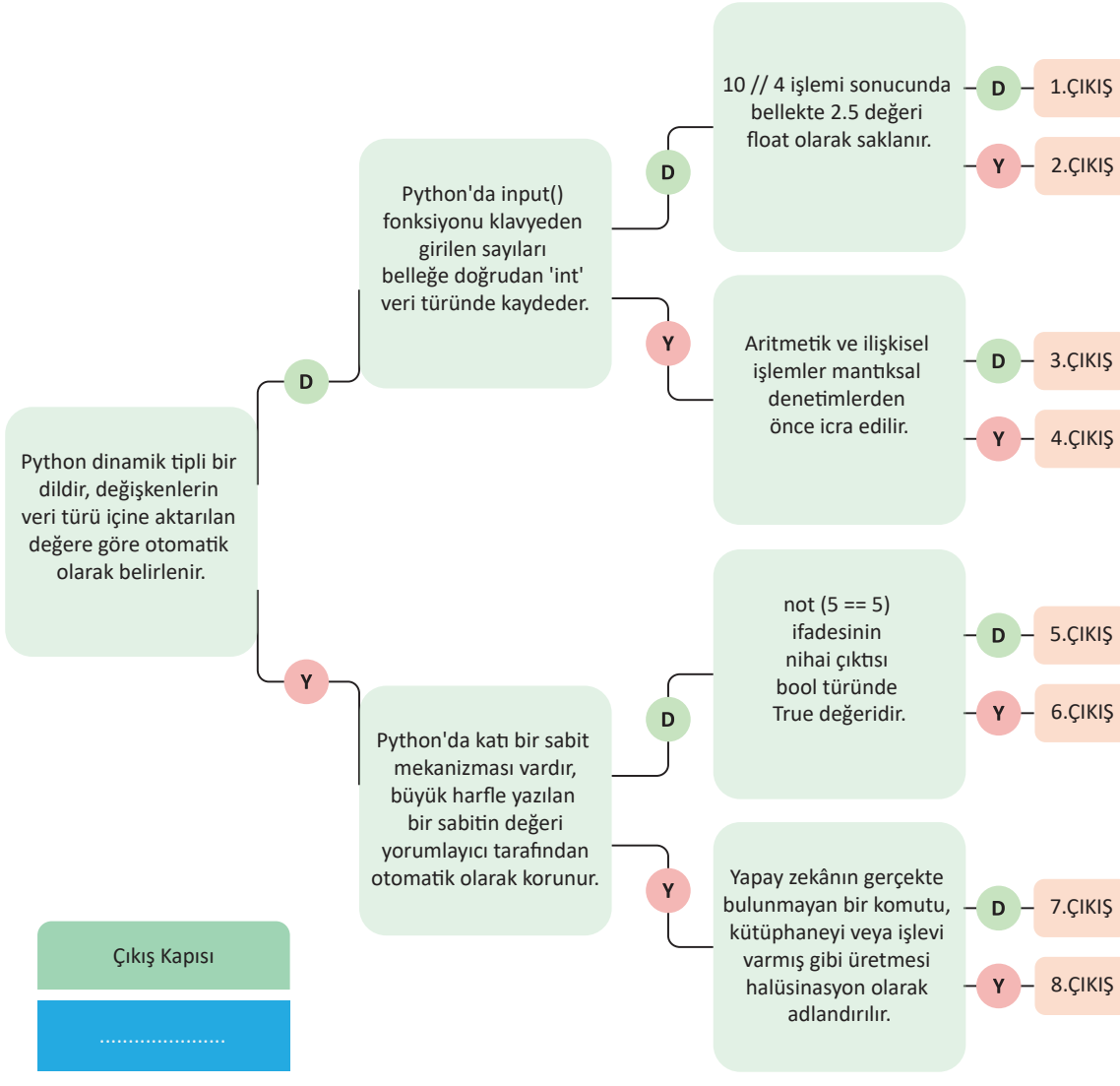
ÇEMBER: Zihnimde Dönen ve Hayatıma Değer Katan Etik Boyut

Yazılım dünyasında kod üretirken yapay zekâ halüsinasyonlarına karşı İnsan Denetimi prensibini işletmenin önemi, dijital depoda dürüstlük / sistematiklik ilkeleri ve yapay zekâyı bir asistan olarak konumlandırırken taşınması gereken mesleki sorumluluklar hakkında **zihnimde yer eden ve hayatım boyunca dikkat edeceğim temel düşünce** şudur:

- .....

ÖLÇME VE DEĞERLENDİRME

A) Aşağıdaki Tanılayıcı Dallanmış Ağaç şemasında yer alan 1. ifadeden (kökten) başlayarak okuma yapınız. Okuduğunuz ifadenin doğru (D) veya yanlış (Y) olduğuna karar vererek ilgili ok yönünde (dallanarak) bir sonraki ifadeye ilerleyiniz. Her yol ayrımında verdiğiniz kararlar sizi en son aşamada 8 farklı çıkış kapısından yalnızca birine ulaştıracaktır. Kendi zihinsel mantık akışınızı takip ederek ulaştığınız Çıkış Kapısı Numarasını sayfa altındaki kutucuğa yazınız.



Çıkış Kapısı

.....

ÖLÇME VE DEĞERLENDİRME

B) Aşağıda 1'den 9'a kadar numaralandırılmış kutucuklardan oluşan yapılandırılmış grid tablosunu ve altındaki soruları dikkatlice inceleyiniz. Her sorunun cevabı veya cevapları tablodaki kutucuklarda yer almaktadır. Bir kutucuğu birden fazla sorunun cevabı olarak kullanabilirsiniz. Soruların altındaki boşluklara, doğru olduğunu düşündüğünüz kutucuk numaralarını yazınız.

1. float	2. and	3. or
4. //	5. str	6. not
7. int	8. %	9. bool

1. Kutucuklardan hangileri Python dilindeki temel veri türlerini temsil eder?

   .....

2. Kutucuklardan hangileri sayısal değerleri temsil etmek amacıyla kullanılan veri türleridir?

   .....

3. Kutucuklardan hangileri birden fazla koşulu birleştirmek veya bir koşulun doğruluk değerini tersine çevirmek için kullanılan mantıksal operatörlerdir?

   .....

4. Kutucuklardan hangileri bölme işlemiyle ilişkili sonuçlar üretmek amacıyla kullanılan aritmetik operatörlerdir?

   .....

5. Kutucuklardan hangileri programlardaki mantıksal karar ve koşul değerlendirme süreçlerinde kullanılan veri türü veya operatörlerdir?

   .....

ÖLÇME VE DEĞERLENDİRME

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Kaynak kodu çalıştırılmadan önce makine koduna veya ara koda dönüştüren yazılım aşağıdakilerden hangisidir?

A) Yorumlayıcı sistemi
B) Derleyici sistemi
C) Etkileşimli kabuk
D) Eklenti mağazası
E) Dosya gezgini

2. Python programlama dilinde bir değişken tanımlanırken tür beyanında bulunulmamasını sağlayan mimari özellik aşağıdakilerden hangisidir?

A) Resmî dil anahtarı
B) Dinamik tipli dil
C) Yılan gösterimi
D) Söz dizimi kuralı
E) Akıllı tamamlama

3. Bir değişkenin bellekte hangi veri türü kalıbıyla saklandığını denetleyen fonksiyon aşağıdakilerden hangisidir?

A) print()
B) input()
C) int()
D) type()
E) float()

4. Bellekte "veri" ve "VERİ" adlarının iki farklı değişken adı olarak değerlendirilmesine yol açan dil özelliği aşağıdakilerden hangisidir?

A) Büyük-küçük harf duyarlılığı
B) Türkçe karakter standardı
C) Geliştirici sözleşmesi
D) Akıllı kod tamamlama
E) Sürüm kontrol sistemi

5. PEP 8 stil önerilerine göre değeri program akışı boyunca değiştirilmemesi gereken sabit verilerin yazım kuralı aşağıdakilerden hangisidir?

A) Tüm harfleri küçük yazılmalıdır.
B) Tüm harfleri büyük yazılmalıdır.
C) Rakam ile başlatılmalıdır.
D) camelCase notasyonuyla yazılmalıdır.
E) Alt çizgiyle bitmelidir.

6. Klavyeden girilen veriyi metin türünde alan yerleşik fonksiyon aşağıdakilerden hangisidir?

A) int()
B) float()
C) str()
D) print()
E) input()

7. Kaynak kod içerisine yazılan "sonuc = 16 / 4" işleminin ardından "sonuc" değişkeninin nihai değeri ve bellek veri türü aşağıdakilerden hangisidir?

A) 4, int
B) 4, str
C) 4.0, float
D) 4.0, bool
E) 4.0, int

8. Bölme sonucunun taban değerini, bir diğer ifadeyle kendisinden büyük olmayan en yakın tam sayıyı üreten operatör aşağıdakilerden hangisidir?

A) *
B) /
C) %
D) //
E) **

9. Bağladığı karşılaştırma ifadelerinden <u>en az</u> bir tanesinin doğru olması durumunda True mantıksal çıktısını üreten operatör aşağıdakilerden hangisidir?

A) and
B) or
C) not
D) ==
E) !=

10. Python yorumlayıcısının standart hiyerarşik öncelik sırasını manuel olarak değiştirerek belirli bir aritmetik alanın daha önce çalıştırılmasını sağlayan sembol hangisidir?

A) Köşeli parantez [ ]
B) Süslü parantez { }
C) Normal parantez ( )
D) Çift tırnak işareti " "
E) Alt çizgi karakteri _

ÖLÇME VE DEĞERLENDİRME

11.
```
asistan_adi = "KODİ"
versiyon = 2
print("Asistan Adı:" + asistan_adi + "Versiyon:" + versiyon)
```

Verilen kod satırları terminalde çalıştırıldığında programın TypeError üretmesine yol açan teknik gerekçe aşağıdakilerden hangisidir?

A) Değişken isimlerinde yılan gösterimi notasyonunun ihlal edilmesi
B) print() fonksiyonu içerisinde f-string altyapısının zorunlu olması
C) Metinsel bir veri ile sayısal bir verinin + operatörü ile birleştirilmeye çalışılması
D) Değişken tanımlanırken veri türü beyanının eksik bırakılması
E) print() fonksiyonunun bitiş parametresi olan end değerinin manuel manipüle edilmesi

12.
```
print("Bilişim", "Teknolojileri", "Alanı", sep="-")
```

Verilen kod satırı Python yorumlayıcısı tarafından çalıştırıldığında terminal ekranına basılacak standart veri çıktısı aşağıdakilerden hangisidir?

A) Bilişim Teknolojileri Alanı
B) Bilişim-Teknolojileri-Alanı
C) BilişimTeknolojileriAlanı-
D) - Bilişim-Teknolojileri-Alanı
E) <class 'str'> Bilişim-Teknolojileri-Alanı

Ç) Aşağıdaki soruların cevaplarını teknik terimleri kullanarak açıklayınız.

1. Aşağıdaki Python kaynak kod bloku çalıştırıldığında terminal ekranına yansıyacak çıktılar nelerdir? Bu çıktıları sırasıyla yazınız. Yorumlayıcının tek bir bellek hücresini atanan değerlere göre nasıl yapılandırdığını gerekçesiyle açıklayınız.

```
asistan_verisi = "Kod-Asistan v0.2"
print(type(asistan_verisi))
asistan_verisi = 100
print(type(asistan_verisi))
asistan_verisi = 99.9
print(type(asistan_verisi))
```

ÖLÇME VE DEĞERLENDİRME

2. **Aşağıdaki kod satırlarındaki hatalar nelerdir? Kodun hatasız, güvenli hâlini yılan gösterimi notasyonuna uygun olarak yeniden yazınız.**

```
1_urun = "Bilgisayar"
ekran-karti-kapasitesi = "16 GB"
def = 1453
```

3. **Aşağıdaki kodlar çalıştırıldığında sonuc1, sonuc2 ve sonuc3 değişkenlerinin bellekte alacağı nihai sayısal değerleri ve bu değerlerin ait olduğu veri türleri nelerdir? Bu bilgileri eşleştirerek yazınız.**

```
sonuc1 = 10 / 4
sonuc2 = 10 // 4
sonuc3 = 10 % 4
```

4. **Kullanıcıdan doğum yılını alarak yaklaşık yaşını hesaplayan ve ekrana yazdıran Python kodu nedir? Yazınız.**

5. **Aşağıdaki kod bloku çalıştırıldığında değişkenlerin üreteceği nihai çıktılar nelerdir? Bu çıktıları sırasıyla yazınız. Python yorumlayıcısının bu satırları anlamlandırırken takip ettiği işlem önceliği kurallarını basamaklar hâlinde açıklayınız.**

```
hesaplama = 10 + 5 * 2
onay = 10 + 5 > 12 and not (3 == 4)
```

6. **Bir programlama ortamının kaynak kodu satır satır çalıştırdığı ve bir satırdaki çalışma zamanı hatasına ulaşıncaya kadar önceki satırları yürüttüğü gözlemlenmiştir. Bu çalışma biçimi derleyici ve yorumlayıcı yaklaşımlarından hangisine daha çok benzemektedir? Gerekçesiyle açıklayınız.**

ÖLÇME VE DEĞERLENDİRME

7. **input() fonksiyonunun program akışında oluşturabileceği veri tipi kaynaklı riskleri açıklayınız. Kullanıcıdan alınan verilerin doğru veri türüyle kullanılabilmesi için hangi fonksiyonlar kullanılmalıdır? Gerekçesiyle açıklayınız.**

8. **Yapay zekâ halüsinasyonu ve İnsan Denetimi prensibi nedir? Açıklayınız.**

9. **Kullanıcıdan adını ve o gün denetlediği kod adedini tam sayı olarak dinamik şekilde alınız. Alınan bu verilerle terminal ekranına f-string kullanarak aşağıdaki şekilde yazdıran Python kodu nedir? Bu kodu oluşturarak aşağıya yazınız.**

   "Hoş geldiniz Mustafa. Bugün yapay zekâ ile 5 kod denetlediniz."

10. **Aşağıdaki kod bloku çalıştırıldığında terminal ekranında oluşacak nihai çıktı nedir? Yazınız ve nedenini gerekçelendiriniz.**

```
# Satır 1
print("Sistem başladı...", end=" [BAŞARILI] ")
# Satır 2
print("Veri tabanı aktif.")
```



Öğrenme Birimi Karekodu

3. ÖĞRENME BİRİMİ
KARAR YAPILARI VE DÖNGÜLER

KONULAR

3.1. KOŞUL YAPILARI VE MANTIKSAL OPERATÖRLER
3.2. DÖNGÜ YAPILARI VE LİSTE İŞLEMLERİ
3.3. GRAFİKSEL GÖRSELLEŞTİRME

ANAHTAR KAVRAMLAR

- .append()
- break
- continue
- .ellipse()
- else
- elif
- for
- if
- ilişkisel operatörler
- .insert()
- .lower()
- mantıksal operatörler
- range()
- remove()

BEKLENTİLER

Bu öğrenme biriminde sizden

- koşul yapılarını kullanarak program akışını yönetmeniz
- tekrarlı işlemleri yönetmek için döngü yapılarını ve liste ilişkisini kullanmanız
- algoritma süreçlerini grafik kütüphaneleri ile görselleştirmeniz

beklenmektedir.

BAŞLARKEN

Günlük yaşamda pek çok durumda karar veririz. Hava yağmurluysa şemsiye almak, otobüs kartında yeterli bakiye varsa araca binmek, yeterli değilse bakiye yüklemek bu kararlara örnektir. Peki, tanımlanan koşullara göre program akışını nasıl yönlendirir?

Şimdiye kadar yazdığınız programlar çoğunlukla yukarıdan aşağıya doğru sıralı biçimde çalışıyordu ancak birçok yazılımın, belirlenen koşullara göre farklı kod bloklarını çalıştırması ve gerektiğinde bazı işlemleri tekrar etmesi gerekir. Bu öğrenme biriminde koşul ve döngü yapılarını kullanarak program akışını yöneteceksiniz. Kod-Asistan projeniz de kullanıcı seçimlerine göre farklı işlemleri gerçekleştirebilen etkileşimli bir yapıya kavuşacaktır.

HAZIRLIK ÇALIŞMALARI

1. Günlük hayatta bir kavşağa geldiğinizde hangi yöne gideceğinize nasıl karar verirsiniz? Bu kararı verirken zihninizde ne gibi "eğer... o hâlde..." durumları oluşur? Düşüncelerinizi arkadaşlarınızla paylaşınız.
2. Bir fabrikada aynı üründen binlerce adet üretildiğini düşününüz. İnsanların bu işlemi yaparken yaşayabileceği zorluklar neler olabilir? Bir makinenin aynı işlemi bıkmadan tekrarlayabilmesi işleyişi nasıl etkiler?
3. Verilen anahtar kavramın size neleri çağrıştırdığını düşününüz. Bu kavramla ilgili aklınıza gelen en fazla 5 kelimeyi ilgili kutucuklara yazınız. Ardından bu kelimeleri kullanarak kavram hakkındaki ön bilgilerinizi yansıtan anlamlı bir cümle kurunuz.

Anahtar Kavram: KOŞUL

a)

b)

c)

ç)

d)

İlgili Cümlem:

.....

.....

.....

.....

.....



3.1. KOŞUL YAPILARI VE MANTIKSAL OPERATÖRLER

Programlama dillerinde, belirlenen bir mantıksal koşulun sonucuna [Doğru (True) veya Yanlış (False)] göre program akışının farklı dallara ayrılmasını sağlayan kontrol mekanizmalarına **karar yapıları** denir. Python dilinde bu süreç if, elif ve else anahtar kelimeleri ile yönetilir.

3.1.1. Tek Yönlü Karar Mekanizmaları: if (Eğer) Yapısı

Bir koşulun doğruluğunu denetleyen ilk temel basamaktır. Eğer belirtilen koşulun mantıksal sonucu True (Doğru) ise if bloku altında girintilenmiş olan kod satırları çalıştırılır. Koşul False ise bu blok tamamen atlanır.

```
if kosul_ifadesi:
    gorev_satiri_1
    gorev_satiri_2
ana_akis_satiri
```

Kod blokunda, kosul_ifadesi değeri True ise gorev_satiri_1, gorev_satiri_2 ve ana_akis_satiri çalışır. Eğer kosul_ifadesi değeri False ise yalnızca ana_akis_satiri kodu çalışır.

3.1.2. Dallanma Mantığı ve Girinti Dünyası

Python'da kod bloklarının sınırlarını belirleyen temel özelliklerden biri **girinti (indentation)** mimarisidir. C++, Go, C# gibi dillerde bir kod blokunun hangi koşula veya döngüye ait olduğunu belirtmek için süslü parantezler ({ }) kullanılırken Python dil mimarisinde bu işlem, doğrudan kodun satır başından itibaren bırakılan boşluklar vasıtasıyla gerçekleştirilir.

Python yorumlayıcısı, bir kontrol yapısının (if, elif, else, for ve while) başladığını belirten iki nokta (:) sembolünü gördüğü andan itibaren bir sonraki satırın içe doğru kaydırılmasını (4 boşluk veya 1 Tab tuşu) bekler. Aynı girinti seviyesinde bulunan tüm kod satırları, tek bir mantıksal blok olarak kabul edilir. Python kod bloklarında her girinti düzeyi için dört boşluk kullanılması önerilir. Aynı dosyada Tab ve boşluk karakterleri karıştırılmamalıdır. Girinti kuralının ihlal edilmesi durumunda sistem doğrudan **IndentationError (Girinti Hatası)** vererek akışı durdurur.

```
# Standart Girinti Yapısı
if kosul_ifadesi:
    # Bu satır koşula bağlıdır (4 boşluk içeride).
    görev_satiri_1
    görev_satiri_2
# Bu satır girintiden çıktığı için koşuldan bağımsızdır, her şekilde çalışır.
ana_akis_satiri
```

3.1.3. Çift Yönlü Alternatifler: if-else (Eğer-Değilse) Yapısı

Gerçek dünya senaryolarında koşulun doğru olması durumunda bir kod blokunun, yanlış olması durumunda ise mutlaka başka bir kod blokunun çalıştırılması istenir. Bu tür çift yönlü yol ayrımlarında if ifadesi else (aksi takdirde) anahtar kelimesiyle birlikte yapılandırılır. else ifadesi tek başına bir koşul barındırmaz, if şartının başarısız olduğu tüm senaryoları otomatik olarak kapsayan güvenli bir limandır.

```
# Kullanıcı Rolü Yetki Kontrolü
kullanici_rolu = "Misafir"
if kullanici_rolu == "Yönetici":
   print("Erişim İzni Verildi. Sistem paneli yükleniyor...")
else:
   print("Erişim Engellendi! Bu alanı görmeye yetkiniz yok.")
```

3.1.4. Karşılaştırma ve Mantıksal Operatörlerin Birleşik Kullanımı

Programlama dillerinde kararlar, çoğunlukla tek bir dinamik verinin kontrol edilmesinden çok daha karmaşıktır. Bir sistemin tanımlanan koşullara göre doğru kod bloklarını çalıştırabilmesi ve doğru çıktılar üretebilmesi, birden fazla nesnel şartın aynı anda ve birleşik bir süzgeçten geçirilmesini gerektirir. Günlük hayatta tasarruf bilinciyle yapılan bir alışverişte sadece bütçenin yeterli olması tek başına bir satın alma sebebi değildir; ürünün gerçek bir ihtiyaç olması da (bütçe yeterli **ve** temel ihtiyaç) gözetilir. Benzer şekilde bir indirim kampanyasından faydalanabilmek için öğrenci belgesi **veya** kurumsal üyelik kartı ibraz edilmesi gibi alternatif şartlar aranır. Yazılım literatüründe de bu tarz çoklu koşulları tasarlamak için ilişkisel operatörler (<, >, ==, !=, <=, >=) vasıtasıyla üretilen tekil mantıksal çıktılar, yerleşik mantıksal operatörler (and, or, not) kullanılarak birbirine bağlanır ve tek bir nihai True ya da False değerine indirgenir.

Birleşik ifadelerin doğru sonuçlar üretebilmesi için Python dil mimarisindeki **Genel İşlem Önceliği Hiyerarşisi** kuralları dikkate alınmalıdır. Bilgisayarın karmaşık bir kod satırını çözerken izlediği bu yol, finansal okuryazarlık gerektiren bir alışverişte insan zihninin izlediği sıraya çok benzer. Bir ürünün alınıp alınmayacağına karar verilirken önce cüzdandaki harçlık ile kumbaradaki tasarruf miktarı toplanır. Ardından elde edilen bu toplam bütçe, hedeflenen ürünün raf etiketiyle kıyaslanır. En son aşamada ise paranın yetip yetmediği bilgisi, o ürünün anlık bir istek mi yoksa gerçek bir ihtiyaç mı olduğu sorusuyla birleştirilerek nihai harcama kararı verilir. Karmaşık bir kod satırı değerlendirilirken Python yorumlayıcısı da bu örnekteki gibi sırasıyla şu adımları izler:

1. **Aritmetik Operatörler**: Önce matematiksel hesaplamalar (*, /, +, -) tamamlanır.
2. **İlişkisel (Karşılaştırma) Operatörler**: Çıkan sayısal sonuçlar birbirleriyle kıyaslanır (==, >, <= vb.) ve ham True / False değerleri üretilir.
3. **Mantıksal Operatörler**: Üretilen mantıksal değerler kendi içinde önce not, sonra and ve en son or hiyerarşisiyle işlenerek tek bir karara bağlanır.

Bu hiyerarşinin kavranması ve kod yazarken karmaşık bölümlerin parantez ( ) ile gruplandırılarak güvence altına alınması, programın yanlış hesaplamalar yapmasının (mantıksal hataların) önüne geçmek için uygulanan en temel yazılım standartlarından biridir.

1. ÖRNEK

KARARLI SEPET VE İNDİRİM HESAPLAMA MODÜLÜ

Kod-Asistan projesine kullanıcıların sepet tutarlarını ve indirim haklarını hesaplayan akıllı bir finansal katman eklenecektir. Uygulamanın kampanya kuralları şu birleşik mantığa göre çalışmaktadır:

- Bir kullanıcının indirim alabilmesi için öncelikle kupon kodunun aktif olması ve kullanıcının yaşının 18'den büyük ya da eşit olması veya toplam sepet tutarının 500 TL'den fazla olması gerekir.
- Eğer bu birleşik şartlar sağlanıyorsa sepet tutarına %10 indirim uygulanacak, şartlar sağlanmıyorsa sepet tutarı aynen korunacaktır.

Örnek Kod Bloku

```
sepet_tutari = float(input("Lütfen mevcut sepet tutarını giriniz (TL): "))
kullanici_yasi = int(input("Lütfen yaşınızı giriniz: "))
# Kampanya parametresinin mantıksal (boolean) olarak tanımlanması
kupon_kod_aktif_mi = True
# BİRLEŞİK OPERATÖR KULLANIMI VE İŞLEM ÖNCELİĞİ ALANI
# Önce parantez içi (or) hesaplanır, çıkan sonuç kupon aktifliği ile (and) bağlanır.
indirim_onaylandi_mi = kupon_kod_aktif_mi and (kullanici_yasi >= 18 or sepet_tutari >
500)
print("\n--- Mantıksal Analiz Sonucu ---")
print(f"Kupon Durumu: {kupon_kod_aktif_mi}")
print(f"Birleşik Şart Sağlandı mı?: {indirim_onaylandi_mi}")
print("--------------------------------")
# Karar mekanizmasının işletilmesi
if indirim_onaylandi_mi:
    indirim_miktari = sepet_tutari * 0.10
    odenecek_tutar = sepet_tutari - indirim_miktari
    print(f"Tebrikler! Kampanya şartlarını sağladınız.")
    print(f"Uygulanan İndirim: {indirim_miktari:.2f} TL")
    print(f"Net Ödenecek Tutar: {odenecek_tutar:.2f} TL")
else:
    print(f"Maalesef kampanya şartları sağlanamadı.")
    print(f"Net Ödenecek Tutar: {sepet_tutari:.2f} TL")
```

Kod Gözden Geçirme (Code Review) Çıkarımı

- Kodda (kullanici_yasi >= 18 or sepet_tutari > 500) blokunun parantez içine alınmasıyla varsayılan işlem önceliği hiyerarşisi yazılımcı tarafından değiştirilmiş ve Python yorumlayıcısının öncelikle bu şartları hesaplaması sağlanmıştır.
- f-string, değişkenlerin metin içine okunabilir ve kolay biçimlendirilebilir şekilde yerleştirilmesini sağlar.

3.1.5. Çok Kriterli Karar Mekanizmaları: if-elif-else Merdiveni

Günlük hayatta ve karmaşık yazılım mimarilerinde kararlar her zaman **Evet / Hayır** gibi iki seçenekten ibaret değildir. Çoğu zaman birbirine bağlı birden fazla koşulun sırayla kontrol edilmesi ve sadece doğru olan yolun seçilmesi gerekir. Python programlama dilinde bu tür çok kriterli karar süreçlerini yönetmek için if ve else ifadelerinin arasına **aksi takdirde eğer** anlamına gelen elif (else-if kısaltması) anahtar kelimesi yerleştirilir.

if-elif-else yapısı yukarıdan aşağıya doğru doğrusal bir hiyerarşiyle çalışır. Python yorumlayıcısı ilk karşılaştığı if koşulunu test eder, eğer bu koşul doğru ise ilgili kod blokunu çalıştırır ve merdivenin geri kalan tüm basamaklarını (elif ve else bloklarını) işletmeden doğrudan kontrol yapısından çıkar. Eğer ilk koşul yanlış ise sırasıyla aşağıdaki elif basamaklarını kontrol etmeye başlar. Koşulların hiçbirinin sağlanmadığı senaryoda ise en altta yer alan ve adeta güvenli bir liman olan else bloku devreye girer.

```
# if-elif-else Merdiven Yapısı Şablonu
if birinci_kosul:
    # Sadece birinci koşul True ise çalışır.
    blok_1
elif ikinci_kosul:
    # Birinci koşul False VE ikinci koşul True ise çalışır.
    blok_2
elif ucuncu_kosul:
    # Önceki koşullar False VE üçüncü koşul True ise çalışır.
    blok_3
else:
    # Yukarıdaki tüm koşullar False ise mutlaka çalışır.
    varsayilan_blok
```

if-elif-else merdiveni kurgulanırken koşulların yazım sırası programın mantıksal kararlılığı açısından kritik bir öneme sahiptir. Yorumlayıcı ilk doğru gördüğü blokta aramayı sonlandırdığı için kapsayıcılığı yüksek olan koşulların alt basamaklara, daha özel veya dar kapsamlı koşulların ise üst basamaklara yazılması gerekir. Aksi takdirde bazı kod blokları hiçbir zaman çalışmayarak **ölü kod (dead code)** hâline gelebilir.

3.1.6. İç İçe Koşul Yapıları

Yazılım dünyasında bazı problemlerde bir koşulun kontrol edilebilmesi, kendisinden önceki başka bir koşulun başarıyla gerçekleşmiş olmasına bağlı olabilir. Bir kontrol yapısının içine (girinti seviyesini bir kademe daha artırarak) yeni bir kontrol yapısı yerleştirilmesine **iç içe (nested) koşul yapısı** adı verilir.

Günlük hayatta ehliyet alma süreci, bu mimariye verilebilecek somut örneklerden biridir. Bir adayın sürücü belgesi alabilmesi için öncelikle 18 yaşını doldurmuş olması şartı aranır. Bu ilk ve dış koşul sağlanmazsa diğer aşamalara geçilmesine izin verilmez ve süreç doğrudan sonlandırılır. Yaş şartı başarıyla sağlandığında ise adayın karşısına **yazılı sınavı geçme** şeklinde yeni bir iç koşul çıkar. Yazılı sınavı geçenler için de **direksiyon sınavında başarılı olma** gibi daha içeride yer alan başka bir koşul devreye girer. Bu ardışık ve tamamen birbirine bağımlı doğrulama aşamalarında olduğu gibi programlama dillerinde de bir kod blokunun çalıştırılması, dıştaki ana koşulun Doğru (True) değerini üretmesine ve ardından içeriye doğru girintilenmiş diğer alt koşulların sırayla sağlanmasına bağlıdır. Bu hiyerarşik yapı, yazılımların gerçek yaşamdaki gibi güvenli ve aşamalı kararlar almasını mümkün kılar.

2. ÖRNEK

İÇ İÇE KARAR YAPILARI VE İKİ AŞAMALI YETKİLENDİRME

Kod-Asistan projesinin siber güvenlik katmanında, sisteme yetkisiz erişimleri engellemek için iki aşamalı bir kimlik doğrulama protokolü tasarlanmıştır. Kullanıcının sistemi açabilmesi için önce metinsel sistem parolasını, ardından telefonuna gelen sayısal doğrulama kodunu girmesi gereklidir. Şifrelerin doğru veya yanlış girilmesi durumlarında sistemin kendi içinde farklı yollara saparak (karar vererek) üreteceği tepkiler test edilecektir.

Örnek Kod Bloku

```
# Siber Güvenlik Katmanı: İki Aşamalı Yetkilendirme Protokolü
sistem_parolasi = "Fatih1453"
dogrulama_kodu = 5532

girilen_parola = input("Sistem parolasını giriniz: ")

if girilen_parola == sistem_parolasi:
   print("1. Aşama Başarılı: Parola Doğrulandı.")

 # İç içe geçmiş ikinci kontrol katmanı (Girinti seviyesi 8 boşluğa çıktı.)
   girilen_kod = int(input("Telefonunuza gelen 4 haneli doğrulama kodunu giriniz: "))

   if girilen_kod == dogrulama_kodu:
      print("2. Aşama Başarılı: Kimlik Doğrulandı. Giriş Yapıldı.")
   else:
      print(" Doğrulama kodu hatalı. 2. Aşama Kod Hatası! Erişim Engellendi.")
else:
   print("Erişim Engellendi: Hatalı Sistem Parolası!")
```

Kod Gözden Geçirme (Code Review) Çıkarımı

- Python dilinde karar yapılarının hangi kapsama ait olduğunu belirleyen yapı girinti seviyesidir.
- İç içe geçmiş bir güvenlik katmanı oluşturulurken kodun daha içeriden (örneğin 8 boşlukla) başlatılması, yorumlayıcıya **Bu şart sadece ilk şart sağlanırsa geçerlidir.** mesajını verir.
- Sayısal kod alınırken int(input(...)) kullanılarak tür dönüşümü yapılması, tür uyumsuzluğundan kaynaklanabilecek TypeError çökme risklerini başarıyla engeller.

1. UYGULAMA

Uygulamanın Adı: ÇOKLU KARAR YAPILARI VE AKILLI MENÜ YÖNETİMİ

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: Çoklu karar yapılarını kullanarak kullanıcı etkileşimli bir akıllı menü tasarlamak, değişkenleri kurallara uygun adlandırmak, mantıksal karşılaştırmalar yapmak ve koddaki hataları ayıklamak

Problem Senaryosu: Kod-Asistan projesinin v0.3 sürümünde kullanıcıya farklı işlevler sunan **Akıllı Menü Sistemi** tasarlanmıştır. Kullanıcının terminal üzerinden yapacağı sayısal seçime (1, 2 veya 3) göre sistemin ilgili modülleri (Siber Güvenlik Testi, Vücut Kitle İndeksi Hesaplayıcı veya Çıkış) tetiklemesi, menü dışı hatalı seçimlerde ise programın çökmeden güvenli bir uyarı vererek işlemi sonlandırması hedeflenmektedir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Kod Dosyasının Hazırlanması ve İncelenmesi

a) Kod-Asistan v0.3 projeniz için yeni bir temiz kod dosyası oluşturarak adını akilli_menu.py olarak kaydediniz.

b) Verilen ve bir mantık hatası (bug) barındıran aşağıdaki menü taslak kodunu editörünüze yazınız.

```
print("=== KOD-ASİSTAN SİSTEMİNE HOŞ GELDİNİZ ===")
kullanici_adi = input("Lütfen adınızı giriniz: ")
print(f"Merhaba {kullanici_adi}. Giriş başarılı. Lütfen yapmak istediğiniz işlemi seçiniz:\n")
# Menü Standartlarının Sunulması
print("1 - Siber Güvenlik ve Kimlik Doğrulama Testi")
print("2 - Kararlı Vücut Kitle İndeksi Hesaplayıcı")
print("3 - Sistem Sürüm Bilgisi ve Çıkış")
secim = input("\nSeçiminiz (1/2/3): ")
if secim == "1":
    print("\n[SİBER GÜVENLİK MODÜLÜ AKTİF]")
    Kullanici_turu = input("Kullanıcı türünüzü giriniz (Yönetici/Misafir): ")
    if kullanici_turu == "Yönetici":
        print("Tam yetki sağlandı. Tüm kaynak kodlar erişime açık.")
    elif kullanici_turu == "Misafir":
        print("Kısıtlı yetki sağlandı. Sadece okuma modundasınız.")
    else:
        print("Tanımsız Rol! Güvenlik protokolü devreye girdi.")

elif secim == "2":
    print("\n[VÜCUT KİTLE İNDEKSİ HESAPLAYICI AKTİF]")
    kilo = float(input("Kilonuzu giriniz (kg): "))
    boy = float(input("Boyunuzu giriniz (Örn: 1.75): "))
    vki = kilo / (boy ** 2)
```

1. UYGULAMA

```
    # f-string ile biçimlendirilmiş çıktı
    print(f"\nHesaplanan İndeks Değeriniz: {vki:.2f}")
    # VKİ Sonucunu Çok Kriterli Değerlendirme
    if vki < 18.5:
        print("Durum: Standart değerlerin altındasınız.")
    elif 18.5 <= vki < 25:
        print("Durum: Tebrikler, standart değerdesiniz.")
    else:
        print("Durum: Standart değerlerin üzerindesiniz.")
elif secim == "3":
    print(f"\nSistemden çıkılıyor... Güvenli günler dileriz {kullanici_adi}.")
    print("Mevcut Proje Deposu: GitHub v0.3 - Kararlı Sürüm.")
else:
    print("\nHatalı Seçim! Lütfen menüde belirtilen rakamlardan (1, 2 veya 3) birini giriniz.")
```

2. Aşama: Kodun Test Edilmesi ve Çalışma Zamanı Hatası (Runtime Error)

a) Sağ üst köşedeki Run Python File butonuna tıklayarak terminal ekranında kodu çalıştırınız.

b) Sistem sizden menü seçimi istediğinde **1** tuşuna basarak siber güvenlik modülüne giriş yapınız.

c) Kullanıcı türü sorulduğunda **Yönetici** yazarak onaylayınız.

ç) Bu aşamada programın NameError: name 'kullanici_turu' is not defined (tanımlanmamış değişken hatası) hatası vererek çöktüğünü gözlemleyiniz.

3. Aşama: Hata Avcılığı ve Güvenli Kod Revizyonu

a) Python dilinin **büyük-küçük harf duyarlılığına (case-sensitivity)** sahip olduğunu hatırlayarak 1. seçimdeki kod blokunu inceleyiniz.

b) Kodu düzelterek daha güvenli hâle getiriniz.

c) Kodu yeniden çalıştırarak 2 ve 3 numaralı menü seçimlerini de test ediniz ve elif-else yapılarının nasıl yönlendiğini deneyimleyiniz.

4. Aşama: Projenin Dijital Depoya Yüklenmesi

a) Hatasız şekilde çalışan kod dosyanızı tarayıcı üzerinden bulut deposuna **Upload files (Dosyaları yükle)** adımıyla ekleyiniz.

b) Sürüm notu (Commit text) alanına **Akıllı menü yapısı if-elif-else bloklarıyla eklendi ve harf duyarlılığı hatası giderildi.** yazarak deponuzu güvenli bir şekilde güncelleyiniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme formuna ulaşmak için karekodu okutunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

2. UYGULAMA

Uygulamanın Adı: AKILLI GÜVENLİK KAPISI VE VERİ STANDARDİZASYONU

Uygulama Süresi: 20 Dakika

Problem Tanımı: Kod-Asistan projesine, kurumsal bir AR-GE merkezinin girişinde çalışacak **Akıllı Güvenlik Kapısı** kontrol modülü tasarlanacaktır. Sistem, personelden dinamik olarak **Rol** ve **Güvenlik Derecesi** şeklinde talep edecektir. Kullanıcıların yapabileceği olası yazım hatalarına (örneğin YÖNETİCİ, yonetici, Yonetici vb.) karşı güvenli geçiş için sistemin şu kuralları işletmesi gerekir:

- Kullanıcının gireceği rol bilgisi standart küçük harfe dönüştürülmelidir.
- Eğer kullanıcının rolü **yonetici** ise güvenlik derecesine bakılmaksızın ekrana "Erişim izni verildi. Hoş Geldiniz Sayın Yönetici." basılmalıdır.
- Eğer kullanıcının rolü **personel** ise sistem iç içe bir kontrolle güvenlik derecesini sorgulamalıdır. Güvenlik derecesi 3 veya daha yüksekse "Erişim izni verildi. Laboratuvar alanına geçebilirsiniz.", 3'ten küçükse "Erişim engellendi. Yetersiz güvenlik derecesi!" uyarısı vermelidir.
- Eğer bu iki rol dışında geçersiz bir veri girilirse sistem "Hatalı rol tanımlaması! Güvenlik protokolü devreye girdi." çıktısını üretmelidir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulama Adımları

a) Kod editöründe kod_asistanim çalışma klasörünüz içinde guvenlik_kapisi.py adında yeni bir Python kaynak kod dosyası oluşturunuz.

b) Kullanıcıdan rol bilgisini ve güvenlik derecesini dinamik olarak alınız. Tip dönüşümü tuzağına düşmeyiniz.

c) Kullanıcının girdiği rol bilgisinin küçük harfe dönüştürülmüş hâlini bir değişkene atayınız.

ç) if-elif-else merdivenini inşa ediniz.

d) Kodunuzu terminalde farklı harf kombinasyonları (Örneğin PeRsOnEl, YONETICI) ve farklı sayılar girerek test ediniz.

e) Kararlı çalışan kod dosyanızı dijital deponuza göndererek "Akıllı güvenlik kapısı modülü iç içe koşullar ve veri temizleme mimarisiyle tamamlandı." notuyla sürüm kontrolünü sağlayınız.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Dereceli puanlama anahtarı formuna ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

3.1.7. Kararlı Veri Kontrolü: Büyük / Küçük Harf Duyarlılığı

Bilgisayarlar, kullanıcıların girdilerini işlerken karakterlerin biçimsel özelliklerine karşı son derece katıdır. Python programlama dili büyük / küçük harf duyarlılığına sahip bir mimariyle çalışır. Bu durum, bellek düzeyinde "A" karakteri ile "a" karakterinin tamamen farklı iki veri olarak kabul edilmesi anlamına gelir.

Kullanıcılardan metinsel veri alırken harf duyarlılığından kaynaklanan mantıksal hataları engellemek ve programın kararlılığını korumak için string metotlarından yararlanılır. Kullanıcının klavyeden metni nasıl yazdığından (hepsi büyük, hepsi küçük veya karışık) bağımsız olarak veriyi standart bir forma dönüştürmek amacıyla en sık kullanılan iki metot şudur:

.lower() Metodu: Uygulandığı metinsel ifadedeki tüm büyük harfleri küçük harfe dönüştürür.

.upper() Metodu: Uygulandığı metinsel ifadedeki tüm küçük harfleri büyük harfe dönüştürür.

3. ÖRNEK

HARF DUYARLILIĞI ÇÖZÜMÜ

Kullanıcı, bir menü seçeneğinde veya onay ekranında Evet cevabını vermek için farklı biçimler kullanmaktadır.

Uygulama Süresi: 20 Dakika

Örnek Kod Bloku

```
# Standart dışı girdi senaryosu
girdi = "EvEt"

# Yöntem A: Metot kullanılmayan durum (Mantıksal Hata Riski)
if girdi == "evet":
   print("İşlem onaylandı.") # Bu blok çalışmaz, çünkü "EvEt" == "evet" sonucu False'tur.

# Yöntem B: Kararlı Dönüşüm Yapılmış Durum (Önerilen Standart)
temiz_girdi = girdi.lower() # "EvEt" verisi tamamen küçük harfe dönüştürülerek "evet" olur.

if temiz_girdi == "evet":
   print("İşlem başarıyla onaylandı.") # Bu blok kararlı bir şekilde çalışır.
```

Kod Analizi

- Aynı harflerden oluşsa dahi "EvEt" ile "evet" karakter dizileri bilgisayar belleğinde tamamen farklı verileri temsil eder.
- Profesyonel yazılım projelerinde son kullanıcıdan gelen girdiler (büyük / küçük harf karışık, boşluklu vb.) her zaman öngörülemez kabul edilir. Yazılımın dayanıklılığını korumak ve if-else karar yapılarının doğru yönlenmesini sağlamak için alınan metinsel verilerin .lower() gibi dönüştürücü metotlarla arka planda standart bir formata getirilmesi yaygın bir veri normalleştirme uygulamasıdır. Bu adım, programı kullanıcı hatalarına karşı kararlı hâle getirir.



Bilgi Kutusu

Veri Standardizasyonu ve Kod Okunabilirliği

Yazılım geliştirme süreçlerinde her bir elif basamağında kullanıcının girebileceği harf kombinasyonlarını tek tek tahmin edip or operatörü ile bağlamaya çalışmak (örneğin if cevap == "E" or cevap == "e":) kod tabanını karmaşıklaştırır ve okunabilirliği düşürür. Sistematik bir programcı, veriyi kontrol yapısına sunmadan hemen önce giriş aşamasında .lower() veya .upper() metotlarıyla standardize eder. Veriyi tek bir kurumsal formata indirgemek hem kodun satır sayısını azaltır hem de mantıksal açıkların oluşmasını erkenden engeller.

3.2. DÖNGÜ YAPILARI VE LİSTE İŞLEMLERİ

Bilgisayarlar, tanımlanan işlemleri yüksek hızla ve tekrar tekrar gerçekleştirebilir. Ancak sonuçların doğruluğu algoritmanın, kodun ve verilerin doğruluğuna bağlıdır. Gerçek dünya senaryolarında belirli bir işlem kümesinin, bir kurala bağlı olarak ardı ardına yeniden çalıştırılması gerekir. Kod bloklarının belirli şartlar çerçevesinde dinamik olarak tekrarlanmasını sağlayan bu kontrol yapılarına **döngü mekanizmaları (iterative structures)** adı verilir. Python dil mimarisinde tekrarlı süreçleri optimize etmek amacıyla iki temel döngü yapısı kullanılır.

Günlük hayatta bir kargo dağıtım merkezinde paketlerin araca yüklenme süreci, bu iki farklı döngü mimarisine verilebilecek somut örneklerden biridir. Bir görevliye önceden belirlenmiş 50 adet koli listesi verilirse görevli, tam 50 kez taşıma işlemi yapar ve süreci tamamlar. Bilgisayar bilimlerinde de tıpkı bu örnekteki gibi tekrar sayısının veya işlenecek veri miktarının başlangıçta açıkça bilindiği durumlarda **for döngüsü** tercih edilir. Diğer taraftan görevliye belirli bir liste verilmediğinde taşıma işlemi, **kargo aracının kasası tamamen dolana kadar** devam eder. Bu senaryoda yükleme işleminin tam olarak kaç tur süreceği baştan bilinmese de sürecin devamlılığı **kasada boş yer bulunması** şartına bağlanmıştır. Tekrar sayısının belirsiz olduğu ve sürecin tamamen anlık bir mantıksal koşula bağlı olarak sürdürüldüğü bu durumlarda ise **while döngüsü** kullanılır. Bu ayrımın kavranması, algoritmaların problem çözme sürecinde doğru aracın seçilmesi ve sistemin sonsuz döngüye girmeden kararlı bir şekilde kurgulanması açısından önemlidir.

3.2.1. Belirli Sayıda Tekrar: for Döngüsü ve Sıralı Veri Dünyası

Bir kod blokunun tam olarak kaç kez çalıştırılacağının başlangıçta net olarak öngörüldüğü senaryolarda for döngüsü tercih edilir. for döngüsü; metinsel ifadeler, listeler (list), belirli sayı aralıkları gibi sıralı bir veri yapısının (iterable) her bir elemanını sırayla döngü değişkenine atarak döngü blokunu otomatik olarak tetikler.

```
# for Döngüsü Söz Dizimi Şablonu
for dongu_degiskeni in sirali_veri_yigini:
    # Her bir eleman için çalıştırılacak blok (4 boşluk girinti)
    gorev_satiri_1
```

4. ÖRNEK

for DÖNGÜSÜYLE METİNSEL İFADELERİ BÜYÜK HARFE DÖNÜŞTÜRME

Uygulama Süresi: 20 Dakika

Kod-Asistan projesinin metin analizi modülünde analiz edilecek kelimenin dışarıdan kullanıcı tarafından belirlenmesi istenmektedir. Ayrıca sisteme girilen kelimedeki harflerin büyüklük / küçüklük durumlarına bakılmaksızın terminal ekranındaki raporlamanın tamamen büyük harflerle (standartlaştırılarak) ve görsel olarak daha zengin bir biçimde kullanıcıya sunulması hedeflenmektedir.

Örnek Kod Bloku

```
# Kullanıcıdan dinamik ve kararlı veri girişinin alınması
kullanici_metni = input("Lütfen analiz edilecek metni giriniz: ")
print("\n=== GELİŞMİŞ KARAKTER ANALİZ RAPORU ===")
# for döngüsü ile girilen metnin her bir karakterinin taranması
for harf in kullanici_metni:
    # .upper() metodu ile taranan karakterin büyük harfe dönüştürülmesi
    buyuk_harf = harf.upper()
    # F-String teknolojisi ile daha zengin ve formatlı bir çıktı üretilmesi
    print(f"Tespit Edilen Karakter -> [ {buyuk_harf} ]")
print("=======================================")
```

Terminal Çıktısı

```
Lütfen analiz edilecek metni giriniz: Fatih
=== GELİŞMİŞ KARAKTER ANALİZ RAPORU ===
Tespit Edilen Karakter -> [ F ]
Tespit Edilen Karakter -> [ A ]
Tespit Edilen Karakter -> [ T ]
Tespit Edilen Karakter -> [ I ]
Tespit Edilen Karakter -> [ H ]
=======================================
```

Kod Analizi

- input() fonksiyonu ile dışarıdan alınan veri belleğe aktarılmış, ardından for döngüsü metnin uzunluğu (5 karakter) kadar bir diğer ifadeyle tam 5 tur (iteration) dönmüştür.
- Döngünün her bir turunda alınan karakter, .upper() metodu süzgecinden geçirilerek küçük olan harfler büyük harflere dönüştürülmüştür.

3.2.2. range() Fonksiyonu

for döngülerinde ardışık sayısal aralıklar oluşturmak ve döngünün tam olarak istenen miktarda dönmesini sağlamak amacıyla yerleşik **range()** (aralık) fonksiyonundan yararlanılır. range() fonksiyonu, başlangıç, bitiş ve adım bilgilerine göre bir sayı aralığını temsil eden yinelenebilir bir range nesnesi oluşturur. Üç farklı parametre kombinasyonuyla yapılandırılır.

Tek Parametreli Kullanım: range(bitis)

Sayaç varsayılan olarak **0'dan başlar** ve girilen bitiş değerine kadar birer birer artarak ilerler. En kritik kural, bitiş değerinin aralığa dâhil edilmemesidir.

range(3) 0, 1 ve 2 değerlerini üretir (Döngü 3 kez döner.).

Çift Parametreli Kullanım: range(baslangic, bitis)

Sayacın sıfırdan değil belirlenen özel bir sayıdan başlaması istendiğinde kullanılır. Başlangıç değerinden başlar ve bitiş değerine (bitiş hariç) kadar birer birer artar.

range(5, 8) 5, 6 ve 7 değerlerini üretir (Döngü 3 kez döner.).

Üç Parametreli Kullanım: range(baslangic, bitis, artis_miktari)

Sayacın kaçar kaçar artacağını veya azalacağını belirleyen üçüncü bir parametre (adım) eklenir. Eğer artış miktarı negatif seçilirse döngü geriye doğru sayma işlemi gerçekleştirir.

range(2, 9, 2) 2, 4, 6 ve 8 değerlerini üretir (Döngü 4 kez döner.).

range(8, 2, -2) 8, 6 ve 4 değerlerini üretir (Döngü 3 kez döner.).

3.2.3. Koşula Bağlı Tekrar: while Döngüsü ve Mantıksal Süreklilik

Döngünün kaç kez çalışacağı başlangıçta bilinmiyorsa tekrar süreci tamamen bir mantıksal koşulun doğruluğuna (True) bağlıysa **while** döngüsü kullanılır. while kelimesi Türkçe literatürde **"-ken, ... olduğu sürece"** anlamına gelir. Koşul ifadesi True değerini ürettiği sürece döngü bloku aralıksız olarak çalışmaya devam eder. Koşul False değerine döndüğü anda döngü sona erer ve program döngüden sonraki satırdan devam eder.

```
# Kurumsal while Döngüsü Söz Dizimi Şablonu
while mantiksal_kosul:
    # Koşul True olduğu sürece çalışacak blok (4 boşluk girinti)
    gorev_satiri
    # Koşulu etkileyen güncelleme adımı
```

3.2.4. Bileşik Atama Operatörleri

Programlama süreçlerinde özellikle döngü mekanizmalarında bir değişkenin mevcut değerinin üzerine yeni bir sayı eklemek, çıkarmak veya bu değeri katlamak çok sık yapılan bir eylemdir. Değişkenin kendi değerini güncellediği bu durumlarda sayac = sayac + 1 şeklinde uzun bir söz dizimi yazmak yerine Python mimarisinde kod okunabilirliğini artıran **bileşik atama operatörleri** (+=, -=, *= ve /=) kullanılır (örneğin sayac += 1, hak -= 1 vb.).

5. ÖRNEK

SAYAÇ YÖNETİMLİ while UYGULAMASI

Uygulama Süresi: 20 Dakika

Kod-Asistan yazılımının sistem loglarını veya belirli bir raporu ekrana ardışık olarak (örneğin 3 kez) basması istenmektedir. Bu işlem, döngünün kaç kez çalışacağının önceden range() gibi bir aralık olarak verilmesiyle değil anlık şekilde güncellenen bir sayacın mantıksal koşula (sayac <= 3) bağlanmasıyla dinamik olarak kontrol edilecektir.

Örnek Kod Bloku

```
# Belirli bir sınıra kadar sayım yapılması
sayac = 1

while sayac <= 3:
    print(f"Mevcut Sayaç Değeri: {sayac}")
    sayac += 1 # Sayaç değeri 1 artırılır (güncelleme adımı).
```

Terminal Çıktısı

Mevcut Sayaç Değeri: 1
Mevcut Sayaç Değeri: 2
Mevcut Sayaç Değeri: 3

Kod Analizi

- Python yorumlayıcısı ilk olarak sayac = 1 değerini alır ve 1 <= 3 (True) olduğu için döngüye girerek ilk çıktıyı basar.
- Ardından sayac, sayac += 1 komutuyla 2'ye ve sonraki turda 3'e yükseltilir.
- sayac 4 değerine ulaştığı anda 4 <= 3 koşulu False ürettiği için döngü kararlı bir şekilde sonlandırılmış ve program ana akışa dönmüştür.

while döngülerinin kurgulanmasında en sık karşılaşılan mantıksal hata, döngü koşulunun hiçbir zaman False konumuna gelememesidir. 5. örnekte sayac += 1 güncelleme satırı unutulsaydı sayac değeri her zaman 1 kalacak, 1 <= 3 koşulu sonsuza kadar True üretecek ve kullanıcı tarafından manuel olarak sonlandırılana kadar program sürekli çalışacaktı. Bu duruma **sonsuz döngü (infinite loop)** adı verilir.

3.2.5. Döngü Kontrol İfadeleri: break ve continue

Döngülerin normal çalışma akışını anlık olarak yönlendirmek, belirli istisnai durumlarda döngüyü erken sonlandırmak veya bir turu atlamak için iki özel anahtar kelime kullanılır.

- **break (döngüyü kır):** Döngü bloku içerisinde break ifadesiyle karşılaşıldığı an, döngü koşulunun ne olduğuna bakılmaksızın döngü tamamen sonlandırılır ve döngüden sonraki ilk satıra geçilir.
- **continue (turu atla):** for döngü bloku içerisinde continue ifadesiyle karşılaşıldığı zaman o turun geri kalan kod satırları çalıştırılmaz, döngü doğrudan bir sonraki turun başına (koşul kontrol adımına) atlar. while döngüsünde ise koşul yeniden değerlendirilir.

6. ÖRNEK

DÖNGÜ KONTROL İFADELERİNİN MANTIĞI

Uygulama Süresi: 20 Dakika

Kod-Asistan yazılımının veri işleme modülünde belirli bir aralıktaki sayıların sırayla işlenip ekrana yazdırılması planlanmıştır. Sistemin belirli bir değere (3 sayısı) denk geldiğinde o adımı atlayıp yoluna devam etmesi, başka kritik bir değere (5 sayısı) ulaştığında ise mevcut döngüyü sonlandırması test edilecektir.

Örnek Kod Bloku

```
# break ve continue Mantığının Gösterimi
for sayi in range(1, 10):
    if sayi == 3:
        continue # 3 sayısı görüldüğünde ekrana yazdırma turu atlanır.
    if sayi == 5:
        break    # 5 sayısı görüldüğünde döngü tamamen sonlandırılır.
    print(f"İşlenen Sayı: {sayi}")
```

Terminal Çıktısı

İşlenen Sayı: 1
İşlenen Sayı: 2
İşlenen Sayı: 4

Kod Analizi

- 1 ve 2 sayıları koşullarda False ürettiği için terminale yazılır.
- Python yorumlayıcısı 3 sayısına geldiğinde if sayi == 3: koşulu True olur ve continue komutu tetiklenir. Yorumlayıcı, o turun geri kalan kod satırlarını (print işlemini) görmezden gelerek o turu atlar ve doğrudan bir sonraki turun başına (4 sayısına) döner.
- 4 sayısı koşullarda False ürettiği için terminale yazılır.
- Python yorumlayıcısı 5 sayısına ulaştığında if sayi == 5: koşulu True olur ve break komutu çalışır. Bu komut, döngü koşulunun ne olduğuna bakılmaksızın (aralık bitmemiş olsa bile) döngüyü acil durdurma butonu gibi tamamen sonlandırır ve programı döngüden çıkarır.

Bilgi Kutusu

Sonsuz Döngülerden Kaçınma ve Algoritmik Disiplin

Tasarlanan bir yazılımın donanım kaynaklarını tüketerek kilitlenmesi, kullanıcı deneyimini doğrudan bozan en kritik kararsızlıklardan biridir. Bir programcı while döngüsü kurgularken, koşulun bir aşamada False olabilmesi veya ulaşılabilir bir break ifadesinin bulanması sağlamalıdır. Kod yazımına başlamadan önce döngü kırılma noktasını kâğıt üzerinde planlamak, kararlı yazılımlar üretmenin sistematik bir kuralıdır.

3. UYGULAMA

Uygulamanın Adı: KULLANICI ETKİLEŞİMİ VE DÖNGÜSEL MENÜ TASARIMI

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: while döngüsü ve if-elif-else yapısını kullanarak kullanıcı etkileşimli bir menü tasarlamak break ifadesiyle çıkış kontrolü sağlamak ve hatalı girişleri yönetmek

Problem Senaryosu: Programın çalışma kuralları şu şekildedir:

- Program başladığında kullanıcıya sürekli dönecek bir menü sunulacaktır.
- Kullanıcı 1 tuşuna basarsa sepet ve indirim hesaplama modülü tetiklenecek, işlem bitince menüye geri dönülecektir.
- Kullanıcı 2 tuşuna basarsa sistem durum raporu basılacak ve işlem bitince menüye geri dönülecektir.
- Kullanıcı 3 tuşuna basarsa döngü break ifadesiyle kararlı şekilde sonlandırılacak ve program kapanacaktır.
- Kullanıcı bu rakamlar dışında hatalı bir veri girerse sistem çökmek yerine kullanıcıyı uyaracak ve menüyü yeniden ekrana getirecektir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Kod Dosyasının Hazırlanması ve İncelenmesi

a) Kod editörünü açarak kod_asistan_dongu.py isimli yeni bir kod dosyası oluşturunuz.

b) Programın sürekli çalışabilmesi için while True: ifadesiyle aktif bir döngü alanı başlatınız.

c) Döngü içerisindeki tüm kod satırlarının **4 boşluk içeride (girintili)** olmasına dikkat ediniz.

ç) Kullanıcının menü seçimini .strip() fonksiyonuyla alınız.

d) Seçim kontrol bloklarını if-elif-else hiyerarşisiyle yapılandırıp 3 numaralı seçenek altında döngüyü sonlandıracak kod anahtarını yerleştiriniz.

3. UYGULAMA

```
print("=== KOD-ASİSTAN AKTİF ÇALIŞMA ORTAMI BAŞLATILDI ===")

# AKTİF DÖNGÜ MOTORU
# Koşul True olduğu sürece program açıkça çıkış istenene kadar kapanmaz.
while True:
    print("\n--- ANA MENÜ ---")
    print("1 - Sepet ve İndirim Hesaplama Katmanı")
    print("2 - Sistem Durum Raporunu Görüntüle")
    print("3 - Güvenli Çıkış")
    print("----------------")

    secim = input("Lütfen yapmak istediğiniz işlemin numarasını giriniz (1/2/3): ") .strip()

    # ÇOK KRİTERLİ DALLANMA VE DÖNGÜ KONTROL ALANI
    if secim == "1":
        print("\n[MODÜL TETİKLENDİ: Sepet Hesaplama]")
        tutar = float(input("Sepet tutarını giriniz (TL): "))
        if tutar > 500:
            print(f"%10 İndirim Uygulandı! Ödenecek Tutar: {tutar * 0.90:.2f} TL")
        else:
            print(f"Net Ödenecek Tutar: {tutar:.2f} TL")

    elif secim == "2":
        print("\n[MODÜL TETİKLENDİ: Örnek Sistem Durumu Raporu]")
        print("Mevcut Sürüm: Kod-Asistan v0.3")
        print("Bellek Durumu: RAM Kararlı | Çekirdek Motor: Aktif")
        print("Depo Entegrasyonu: Bulut Deposu Bağlantısı Hazır.")

    elif secim == "3":
        print("\nGüvenli çıkış protokolü devreye girdi.")
        print("Güvenli günler dileriz!")
        # KONTROL İFADESİ: Döngü burada tamamen kırılır ve program sonlanır.
        break

    else:
        # Kararsız veri girişlerinin yönetilmesi (Hata Engelleyici Katman)
        print("\nHatalı Seçim! Girdiğiniz değer menü standartlarına uymuyor.")
        print("Lütfen yalnızca 1, 2 veya 3 değerlerinden birini giriniz.")
print("\n=== PROGRAM BAŞARIYLA KAPATILDI ===")
```

3. UYGULAMA

2. Aşama: Etkileşimli Deney

Geliştirdiğiniz kodu terminal üzerinde çalıştırarak aşağıdaki adımları sırayla uygulayınız ve programın tepkilerini gözden geçiriniz.

a) **Deney (Kararlılık Testi)**: Menü ekranı geldiğinde klavyeden rastgele harfler (abc) veya geçersiz rakamlar (5, 0) girip Enter tuşuna basınız. Program çöktü mü yoksa size uyarı mesajı verip ana menüyü yeniden karşınıza getirdi mi? Bu kararlı korumayı sağlayan kod bloku hangisidir?

b) **Deney (Süreklilik Testi)**: 1 seçeneğini seçip sepet hesaplamasını tamamlayınız. İşlem bittiğinde programın tamamen kapanmayıp sizi yeniden --- ANA MENÜ --- satırına döndürmesini sağlayan döngüsel mantığı açıklayınız.

c) **Deney (Çıkış Analizi)**: 3 seçeneğini girdiğinizde ekrana en alttaki "=== PROGRAM BAŞARIYLA KAPATILDI ===" satırının basılmasını ve döngünün o an bitmesini sağlayan anahtar kelimenin çalışma prensibini yorumlayınız.

3. Aşama: Kod İyileştirme

a) Mevcut kod yapısında kararsız veri girişleri (hatalı menü seçimleri), if-elif-else merdiveninin en altındaki else bloku ile denetlenmekteydi. Geçersiz girişleri en alttaki else blokuna bırakmak yerine, en başta filtreleyerek continue ile doğrudan döngünün başına yönlendirmek, kodun okunabilirliğini artıran ve iç içe geçmiş (nested) karmaşık yapıları engelleyen bir yöntemdir. Sisteminizin kodunu sadeleştiren çok kriterli dallanma (if-elif) bloklarının hemen öncesine şu kodu ekleyiniz:

```
if secim != "1" and secim != "2" and secim != "3":
    print("\nHatalı Seçim! Girdiğiniz değer menü standartlarına uymuyor.")
    print("Lütfen yalnızca 1, 2 veya 3 değerlerinden birini giriniz.")
    continue
```

b) En alttaki else kod blokunu siliniz.

4. Aşama: Projenin Dijital Depoya Yüklenmesi

a) Koşul ve döngü yapılarını bir arada harmanlayarak tamamen interaktif ve kararlı hâle getirdiğiniz kod_asistan_dongu.py dosyanızı kaydediniz.

b) İnternet tarayıcınızdan dijital depo hesabınıza giriş yaparak kod-asistan-v0.2 deponuzu açınız. **Add file > Upload files** adımlarını izleyerek güncel dosyanızı yükleme alanına sürükleyiniz.

c) Sürüm notu alanına şu açıklamayı yazınız ve değişiklikleri kaydediniz: "Ana gövde, while True döngü motoru içerisine alınarak kararlı bir menü altyapısı oluşturuldu; break ifadesiyle güvenli çıkış protokolü koda entegre edildi (v0.3)."

Değerlendirme: Çalışmanızı, karekodda yer alan Öz Değerlendirme Formu'ndaki ölçütlere göre değerlendiriniz. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Kendi çalışmanızı değerlendirmek için karekodda yer alan Öz Değerlendirme Formu'nu doldurunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

3.2.6. Döngülerin Yazılım Verimliliğindeki Rolü

Yazılım mühendisliğinde algoritmik verimlilik, bir çözümün özellikle yürütme süresi ve bellek kullanımı bakımından kaynak maliyetinin değerlendirilmesidir. Kodun okunabilirliği ve bakım kolaylığı ise ayrıca dikkate alınır. Döngü yapıları, yazılım dünyasında tekrarlı işlemleri yeniden kullanılabilir ve yönetilebilir biçimde ifade etmemizi sağlayan temel kontrol mekanizmalarıdır. Eğer döngüler olmasa binlerce veriyi işlemek veya tekrarlı görevleri yürütmek için aynı kod satırlarını binlerce kez kopyalamak gerekir. Bu durum şu kritik verimsizliklere yol açar:

Kod Şişmesi (Code Bloat): Kaynak kod dosyasının boyutunun gereksiz yere büyümesi ve okunabilirliğinin yok olmasıdır.

Bakım Zorluğu: Kodun tek bir yerinde yapılacak bir değişikliğin, kopyalanan tüm satırlarda tek tek yapılmak zorunda kalınmasıdır (Mantıksal hata riskini artırır.).

Kod Tekrarı ve Yürütme Maliyeti: Python yorumlayıcısının doğrusal satırları okurken harcadığı zamanın ve enerjinin artmasıdır.

Döngüler, bin satırlık bir işi sadece üç dört satıra indirgeyerek hem yazılımcının zamanını korur hem de bilgisayar donanımının algoritmaya ve uygulamaya bağlı olarak daha verimli çalışmasına katkı sağlar.

Günlük hayatta bir okulun mezuniyet töreni için bin öğrenciye davetiye hazırlanması süreci, verimlilik ilkesine verilebilecek somut örneklerden biridir. Davetiyelerin her birinin tek tek elle yazılması; gereksiz bir zaman ve kâğıt kaybına (kod şişmesi) yol açar, yazan kişinin enerjisini tüketir (zaman ve iş gücü kaybı) ve tören saatinde bir değişiklik yapıldığında tüm davetiyelerin tek tek silinip baştan yazılmasını (bakım zorluğu) zorunlu kılar. Bunun yerine tek bir şablon tasarlanıp fotokopi makinesine **Bu şablonu bin kez çoğalt**. talimatının verilmesi, süreci hatasız bir verimliliğe kavuşturur. Yazılım projelerinde de döngüler bu çoğaltma makinesi gibi çalışarak binlerce satırlık tekrarlı bir işi sadece üç dört satıra indirger. Bu sayede hem yazılımcının zamanı korunur hem de bilgisayar donanımının algoritmaya bağlı olarak daha verimli çalışmasına katkı sağlanır.

3.2.7. for ve while Döngüleri Arasındaki Karşılaştırmalı Farklar

Python dilinde her iki döngü yapısı, tekrarlı işlemleri yürütse de çalışma biçimleri açısından birbirlerinden tamamen ayrılırlar. Tablo 3.1'de for ve while döngülerinin karşılaştırılması verilmiştir.

Tablo 3.1: for ve while Döngülerinin Karşılaştırılması

Özellik / Kriter	for Döngüsü	while Döngüsü
Tekrar Sayısı	Bir yinelenebilir yapının elemanları üzerinde dolaşır, eleman sayısının önceden bilinmesi zorunlu değildir.	Bir koşul True olduğu sürece çalışır, tekrar sayısı belirli de belirsiz de olabilir.
Çalışma Koşulu	Sıralı bir nesnenin (range, str vb.) elemanları bitene kadar döngüyü sürdürür.	Mantıksal koşul ifadesi True olduğu sürece döngüyü sürdürür.
Sayaç Yönetimi	Her turda yinelenebilir yapıdan aldığı sıradaki değeri döngü değişkenine atar.	Döngünün sonlanmasını sağlayacak koşul değişimi veya ulaşılabilir bir break bulunmalıdır.
En Sık Kullanım Alanı	Liste taramalarında, veri setlerinde ve sabit sayıda dönmesi gereken sayaçlarda kullanılır.	Aktif menü motorlarında, kullanıcı onay ekranlarında, sonsuz veri akışlarında kullanılır.

4. UYGULAMA

Uygulamanın Adı: SAYI TAHMİN OYUNUYLA DÖNGÜ KONTROLÜ

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: Rastgele sayı üretimi, while döngüsü, break ve continue ifadelerini kullanarak kullanıcı etkileşimli bir tahmin oyunu tasarlamak ve döngü kontrolünü sağlamak

Problem Senaryosu: Kod-Asistan projesine kullanıcının algoritmik mantığı eğlenceli bir yolla deneyimlemesini sağlayacak bir oyun modülü entegre edilecektir. Oyunun çalışma ve döngü kuralları şu şekildedir:

- Sistem, başlangıçta random (rastgele) modülünü kullanarak 1 ila 50 arasında gizli bir tam sayı üretecektir.
- Kullanıcının gizli sayıyı tahmin etmek için toplam 5 hakkı bulunacaktır. Bu durum bir sayaçla takip edilecektir.
- Eğer kullanıcı 1-50 aralığının dışında (örneğin 65 veya -5 gibi) geçersiz bir tahmin yaparsa sistem hakkını eksiltmeyecek, kullanıcıyı uyaracak ve continue komutuyla bu adımı atlayarak yeni bir tahmin isteyecektir.
- Kullanıcı sayıyı doğru tahmin ettiğinde oyunun daha fazla uzamaması için döngü break komutuyla kırılacak ve oyun zaferle bitecektir.
- Kullanıcının hakkı bittiğinde döngü kendiliğinden sona erecek ve gizli sayı ekrana basılacaktır.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Kod Dosyasının Hazırlanması ve İncelenmesi

a) Kod editörünü açarak kod_asistan_oyun.py isimli yeni bir kod dosyası oluşturunuz.

b) Python'un random modülünü devreye almak için kodun en üstüne import random satırını ekleyiniz.

c) while kalan_hak > 0: yapısıyla kullanıcının tahmin hakkı sürdüğü müddetçe aktif kalacak bir döngü motoru kurgulayınız.

ç) Kullanıcının girdiği veriyi int() fonksiyonuyla sayıya dönüştürmeden önce veya hemen sonra aralık kontrolüne (if tahmin < 1 or tahmin > 50:) tabi tutarak continue filtresini yerleştiriniz.

d) Doğru tahmin, yüksek tahmin ve düşük tahmin olasılıklarını if-elif-else merdiveniyle yapılandırıp doğru tahmin altında break anahtarını kullanınız.

4. UYGULAMA

```
import random

print("=== KOD-ASİSTAN: SAYI TAHMİN OYUNU MODÜLÜ AKTİF ===")
print("Sistem 1 ile 50 arasında gizli bir sayı belirledi. Bulmak için 5 hakkınız var!\n")

# 1. AŞAMA: Oyun parametrelerinin ve gizli sayının bellekte hazırlanması
gizli_sayi = random.randint(1, 50)
kalan_hak = 5
tahmin_sayaci = 0

# 2. AŞAMA: Oyun Döngüsünün Başlatılması
while kalan_hak > 0:
    girdi = input(f"Kalan Hakkınız ({kalan_hak}). Tahmininizi giriniz: ").strip()
    tahmin = int(girdi)
    # GEÇERSİZ ARALIK GİRİŞİNİN "CONTINUE" İLE ATLANMASI
    if tahmin < 1 or tahmin > 50:
        print("Uyarı: Tahmininiz 1 ile 50 arasında olmalıdır!")
        print("Sistem kararlılığını korumak için bu adım atlanıyor ve hakkınız eksilmiyor.\n")
        # KONTROL İFADESİ: Aşağıdaki kodları çalıştırmadan doğrudan döngü koşuluna döner.
        continue
    # Geçerli bir tahmin yapıldığı için sayaçları güncelleme adımı
    tahmin_sayaci += 1
    kalan_hak -= 1
    # 3. AŞAMA: ÇOK KRİTERLİ DOĞRULAMA VE DÖNGÜ YÖNETİMİ
    if tahmin == gizli_sayi:
        print(f"\nTebrikler! Gizli sayıyı {tahmin_sayaci}. denemenizde buldunuz.")
        # KONTROL İFADESİ: Koşulun hâlâ hakkı olmasına bakılmaksızın döngüyü anında kırar.
        break
    elif tahmin > gizli_sayi:
        print("İpucu: Daha KÜÇÜK bir sayı girmelisiniz.\n")
    else:
        print("İpucu: Daha BÜYÜK bir sayı girmelisiniz.\n")
# 4. AŞAMA: Oyun Sonu Durum Kontrolü
if kalan_hak == 0 and tahmin != gizli_sayi:
    print("\nMaalesef tahmin hakkınız bitti ve oyun sonlandı.")
    print(f"Sistemin belirlediği gizli sayı: {gizli_sayi} idi. Bir sonraki sefere başarılar!")

print("\n=== OYUN MODÜLÜ KAPATILDI ===")
```

4. UYGULAMA

2. Aşama: Etkileşimli Deney

Yazdığınız oyun kodunu terminalde yürüterek aşağıdaki adımları sırayla simüle ediniz ve program akışının mantıksal kararlarını muhakeme ediniz.

a) **Deney (continue İşlevi):** Oyun başlar başlamaz sırasıyla 99, -10 ve 55 değerlerini giriniz. Programın size uyarı verip Kalan Hakkınız (5) satırını korumasını sağlayan algoritmik mantığı gerekçelendiriniz. continue satırı olmasaydı kalan hakkınız bu hatalı girişlerde ne olurdu?

b) **Deney (Erken Sonlandırma Testi):** İlk tahmininizde ipucunu kullanarak ikinci tahmininizde doğru sayıyı bulmaya çalışınız (Gerekirse kodu test etmek için gizli sayıyı ekrana yazdırabilirsiniz.). Sayıyı doğru tahmin ettiğiniz anda kalan 3 hakkınız olsa bile break ifadesi mevcut döngüyü sonlandırır ve programın döngüden sonraki kod satırlarıyla devam etmesini sağlar. Bu komutun etkisini yorumlayınız.

3. Aşama: Projenin Dijital Depoya Yüklenmesi

a) Döngü kontrolünü sağlayan break ve continue ifadelerini kararlı bir oyun senaryosu üzerinde kodlayarak oluşturduğunuz kod_asistan_oyun.py dosyanızı kaydediniz.

b) Dijital depo hesabınızdaki kod-asistan-v0.2 deponuza giderek **Add file > Upload files** adımlarıyla dosyanızı deponuza yükleyiniz.

c) Sürüm notu alanına kurumsal şeffaflık standartlarına uygun olarak şu notu ekleyiniz: "Asistan projesine random modülü entegre edildi, break ve continue komutlarının işlevlerini somutlaştıran Sayı Tahmin Oyunu katmanı başarıyla eklendi (v0.2)."

Değerlendirme: Çalışmanızı, karekodda yer alan Öz Değerlendirme Formu'ndaki ölçütlere göre değerlendiriniz. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Kendi çalışmanızı değerlendirmek için karekodda yer alan Öz Değerlendirme Formu'nu doldurunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

3.2.8. Liste İşlemleri (Veri Kümeleri ve Döngü Entegrasyonu)

Python'daki temel veri tiplerinde (str, int, float,bool) bir değer, tek bir değişken adıyla temsil edilir. Birbiriyle ilişkili birden fazla değeri topluca yönetmek için liste gibi veri yapılarından yararlanılır. Örneğin sınıftaki 5 öğrencinin adı veya bir sepet uygulamasındaki farklı ürün fiyatları kaydedilmek istendiğinde her biri için ayrı ayrı değişkenler (ogrenci_1, ogrenci_2 vb.) tanımlamak zorunda kalınır. Gerçek dünya problemlerinde ise yan yana duran, birbiriyle ilişkili yüzlerce hatta milyonlarca veriyi tek bir çatı altında düzenli bir şekilde tutma ihtiyacı doğar.

Python programlama dilinde, birden fazla veriyi tek bir değişken ismi altında, belirli bir sıra dâhilinde saklayan dinamik veri yapılarına **liste (list)** adı verilir. Listeler, birden fazla verinin düzenli biçimde saklanmasını ve döngülerle işlenmesini kolaylaştırır ve yazılım kalitesini doğrudan artırır.

Python'da bir liste oluşturulurken veriler **köşeli parantez [ ]** içerisine alınır ve her bir eleman birbirinden **virgül (,)** sembolü ile ayrılır. Listelerin en büyük esnekliği, içerilerinde aynı anda farklı veri türlerini (me-

tin, tam sayı, ondalıklı sayı, boolean vb.) barındırabilmeleridir.

```
# Standart Liste Tanımlama Örnekleri
bos_liste = [ ] # İçinde eleman olmayan boş bir liste
asistan_modulleri = ["Menü", "Hesaplama", "Raporlama"] # str türünde elemanlar
sepet_verileri = [150.50, 45, True, "Fatih"] # Farklı türde elemanlar
```

Bir listenin içerisindeki belirli bir elemana doğrudan ulaşmak, onu güncellemek veya silmek için elemanın listedeki konum numarası kullanılır. Yazılım literatüründe bu konum numarasına indeks (index) adı verilir. Python dil mimarisinde indeksleme (indexing) kuralları şu sistematik temellere dayanır:

Pozitif İndeksleme: Listenin ilk elemanının indeks numarası her zaman 0 (sıfır) ile başlar ve sağa doğru birer birer artarak ilerler.

Negatif İndeksleme: Listenin son elemanına hızlıca erişmek için sağdan sola doğru -1, -2 şeklinde ilerleyen negatif indeks mimarisi kullanılır.

```
# İndeks Erişim Modülü
ogrenciler = ["Ahmet", "Zeynep", "Can", "Elif"]
# Pozitif İndeks ile Erişim
ilk_ogrenci = ogrenciler[0]  # "Ahmet" verisini döndürür.
ucuncu_ogrenci = ogrenciler[2] # "Can" verisini döndürür.
# Negatif İndeks ile Erişim
son_ogrenci = ogrenciler[-1]  # "Elif" verisini döndürür.
```

Olmayan bir indeks numarasına erişmeye çalışmak sistem hatalarına yol açar. Dört elemanlı bir listede 4 veya daha büyük bir pozitif indekse erişilmeye çalışılırsa IndexError oluşur. Python IndexError istisnası üretir hata yakalanmazsa programın yürütülmesi durur. Python yorumlayıcısı bu durumu ekrana IndexError: list index out of range (liste indeksi aralık dışında) hatasıyla yansıtır. Kararlı bir yazılımcı, sınır değerleri daima bu kuralı gözeterek yönetir.

3.2.9. Dinamik Döngü-Liste İlişkisi (Iterating Through a List)

Listelerin gerçek verimliliği, for döngüsüyle sarmal bir şekilde bağlandıklarında ortaya çıkar. for döngüsü, listenin ilk elemanından başlayarak son elemanına kadar tüm değerleri sırayla döngü değişkeninin içine aktarır. Böylece tek tek indeks numarası yazmaya gerek kalmadan listenin boyutu ne olursa olsun tüm elemanlar sırayla, her turda bir eleman olacak şekilde işlenir.



Bilgi Kutusu

Sıfır Tabanlı İndeksleme Tasarımı

Günlük hayatta saymaya 1'den başlarken bilgisayar mimarisinde listelerin 0'dan başladığını unutmak, mantıksal sapmaların en temel kaynağıdır. 10 elemanlı bir veri kümesinin son elemanı 10. indekste değil 9. indekstedir. Programlama yaparken bellek şemalarını zihinde canlandırmak, indeks sınırlarını sistematik olarak kontrol etmek ve veri kaymalarının önüne geçmek kararlı kod üretiminin vazgeçilmez bir parçasıdır.

7. ÖRNEK

DİNAMİK DÖNGÜ-LİSTE İLİŞKİSİ VE VERİ ANALİZİ

Kod-Asistan projesinin sepet hesaplama modülünde, bir kullanıcının seçtiği farklı ürün fiyatları tek bir liste içerisinde tanımlanmıştır. Sistemden, bu listedeki tüm fiyatları döngü yardımıyla sırayla ekrana basması ve ardından her bir fiyatı biriktirerek toplam sepet tutarını kullanıcıya sunması istenmektedir.

Örnek Kod Bloku

```
# Listeyi Döngü Yardımıyla Ekrana Basma
fiyatlar = [120, 250, 480, 90]
toplam = 0

print("=== Sepetteki Ürünlerin Fiyat Listesi ===")
for fiyat in fiyatlar:
    print(f"Ürün Tutarı: {fiyat:.2f} TL")
    toplam += fiyat # Her turda gelen fiyat toplam değişkenine eklenir.

print(f"Toplam Sepet Tutarı: {toplam:.2f} TL")
```

=== Sepetteki Ürünlerin Fiyat Listesi ===
Ürün Tutarı: 120.00 TL
Ürün Tutarı: 250.00 TL
Ürün Tutarı: 480.00 TL
Ürün Tutarı: 90.00 TL
Toplam Sepet Tutarı: 940.00 TL

Kod Analizi

- fiyatlar listesinde 4 eleman olduğu için for döngüsü otomatik olarak 4 tur dönmüştür.
- Python yorumlayıcısı her turda fiyat değişkenine sıradaki ürün tutarını atamış, ardından bu değeri toplam += fiyat komutuyla genel toplama (havuza) dâhil etmiştir.
- Herhangi bir sınır değeri belirtmeye gerek kalmadan tüm liste verileri verimli bir şekilde taranmış ve işlenmiştir.

5. UYGULAMA

Uygulamanın Adı: DİNAMİK LİSTE YÖNETİMİ VE SEPET GÜNCELLEME MODÜLÜ

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: Liste, while, for, break ve continue yapılarını kullanarak dinamik veri girişi yapmak, geçerli verileri listeye ekleyip bu veriler üzerinde aritmetiksel analizler gerçekleştirmek

Problem Senaryosu: Geliştirmekte olduğunuz Kod-Asistan projesinin finansal katmanını daha esnek ve gerçekçi hâle getirmek için kullanıcıların istedikleri adette ürün fiyatını sisteme dinamik olarak ekleyebilecekleri bir sepet mimarisi tasarlanacaktır. Algoritmanın çalışma kuralları şu şekildedir:

- Başlangıçta içi tamamen boş olan bir sepet_listesi bellekte tahsis edilecektir.
- Kullanıcı klavyeden bir sayısal fiyat girdiğinde bu veri listeye eklenecek ve program yeni bir fiyat istemek üzere döngüyü sürdürecektir.
- Eğer kullanıcı negatif veya sıfır değerli geçersiz bir fiyat girerse (fiyat <= 0) sistem kullanıcıyı uyaracak, veriyi listeye eklemeden continue komutuyla doğrudan bir sonraki veri isteme adımına atlayacaktır.
- Kullanıcı fiyat yerine büyük / küçük harf duyarsız olacak şekilde "HESAPLA" komutunu girdiğinde veri giriş döngüsü break ile kırılacaktır.
- Döngü kırıldıktan sonra listenin içinde toplanan tüm fiyatlar bir for döngüsüyle baştan sona taranacak, sepetin toplam tutarı hesaplanarak f-string standardıyla ekrana basılacaktır.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Kod Dosyasının Hazırlanması ve İncelenmesi

a) Kod editörünü açarak kod_asistan_sepet_guncel.py isimli yeni bir kod dosyası oluşturunuz.

b) sepet_listesi = [ ] satırıyla dinamik veri kümesini başlatıp, veri toplama akışı için while True: motorunu devreye alınız.

c) Kullanıcının hem metinsel komut (HESAPLA) hem de sayısal fiyat girebilmesini sağlamak için girdiyi ilk aşamada ham metin olarak alıp .upper().strip() süzgecinden geçiriniz.

ç) Girdi kontrolünü if-elif-else hiyerarşisiyle tasarlayınız. Sayısal dönüşümü sadece girdi komut değilse gerçekleştirerek olası ValueError çökmelerinin önüne geçiniz.

d) Döngü sonrasında listenin elemanlarını for döngüsüyle işleyip nihai finansal çıktıyı terminale basınız.

5. UYGULAMA

```
# Geliştirici: [Öğrenci Adı Soyadı]
print("=== KOD-ASİSTAN: DİNAMİK SEPET YÖNETİM PANELİ ===")
print("NOT: Ürün fiyatlarını sırayla giriniz. Veri girişini bitirmek için 'HESAPLA' yazınız.\n")
# 1. AŞAMA: Bellekte boş dinamik listenin tanımlanması
sepet_listesi = [ ]
# 2. AŞAMA: Kesintisiz Veri Toplama Döngüsü (while)
while True:
    girdi = input("Ürün Fiyatı veya Komut: ").strip()
    # Harf Duyarlılığının Çözülmesi ve Çıkış Protokolü
    if girdi.upper() == "HESAPLA":
        print("Veri girişi sonlandırıldı. Hesaplama katmanına geçiliyor...")
        # KONTROL İFADESİ: Veri giriş döngüsü kırılır.
        break
    # Sayısal Dönüşüm ve Kararsız Veri Giriş Filtresi
    else:
        # Metinsel ifadenin float veri türüne güvenli dönüşümü
        urun_fiyati = float(girdi)
        # Hatalı/Mantıksız Veri Girişinin Filtrelenmesi
        if urun_fiyati <= 0:
            print("Hata: Ürün fiyatı sıfır veya negatif olamaz!")
            print("Bu kararsız veri listeye eklenmeden atlanıyor...\n")
            # KONTROL İFADESİ: Aşağıdaki satırları çalıştırmadan döngünün başına döner.
            continue
        # Geçerli verinin listeye eklenmesi
        sepet_listesi.append(urun_fiyati)
        print(f"{urun_fiyati:.2f} TL sepete eklendi. (Güncel Ürün Adedi: {len(sepet_listesi)})\n")
# 3. AŞAMA: Listenin Döngü Yardımıyla Taranması ve Analiz Edilmesi (for)
print("\n=== FATURA DETAY RAPORU ===")
toplam_tutar = 0
urun_sayaci = 1
for fiyat in sepet_listesi:
    print(f"{urun_sayaci}. Ürün Bedeli: {fiyat:.2f} TL")
    toplam_tutar += fiyat
    urun_sayaci += 1
print("--------------------------")
print(f"Sepetteki Toplam Ürün: {len(sepet_listesi)} Adet")
print(f"Net Ödenecek Tutar: {toplam_tutar:.2f} TL")
print("==========================")
```

5. UYGULAMA

2. Aşama: Etkileşimli Deney

Yazdığınız kodu terminalde yürüterek şu adımları test ediniz ve bellek şemalarını muhakeme ediniz.

a) **Deney (Hata Atlama Testi):** Sırasıyla 150, -40, 200 ve 0 değerlerini giriniz. Programın negatif ve sıfır değerlerinde uyarı verip listenin yapısını korumasını sağlayan mantıksal mekanizmayı açıklayınız.

b) **Deney (Komut Kararlılığı):** Girişi sonlandırmak için karışık harflerle HeSaPla yazınız. Metinsel metodun bu esnekliği nasıl sağladığını kod satırı üzerinden analiz ediniz.

c) **Deney (Açık Uçlu Sorgu):** Fatura detay raporundaki urun_sayaci değişkeni neden listedeki indeksler gibi 0 yerine 1 değerinden başlatılmıştır? Kullanıcı deneyimi açısından gerekçelendiriniz.

3. Aşama: Projenin Dijital Depoya Yüklenmesi

a) Listeleri, iki farklı döngü türünü ve kontrol ifadelerini tek bir çatı altında birleştirerek kararlı hâle getirdiğiniz kod_asistan_sepet_guncel.py dosyanızı kaydediniz.

b) Dijital depo hesabınızdaki kod-asistan-v0.2 deponuza giderek **Add file > Upload files** adımlarıyla dosyanızı deponuza yükleyiniz.

c) Sürüm notu alanına, kurumsal şeffaflık standartlarına uygun olarak şu notu ekleyiniz: "Sepet mimarisi dinamik liste yapısına kavuşturuldu; while-for döngüleri, break ve continue ifadeleriyle kararlı veri doğrulama katmanı Kod-Asistan projesine eklendi (v0.3)."

Değerlendirme: Çalışmanızı, karekodda yer alan Öz Değerlendirme Formu'ndaki ölçütlere göre değerlendiriniz. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Kendi çalışmanızı değerlendirmek için karekodda yer alan Öz Değerlendirme Formu'nu doldurunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

3.2.10. Liste Metotları ve Bellek Davranışları

Python programlama dilinde listeler, dinamik genişleme ve daralma yeteneğine sahip esnek veri yapılarıdır. Python listeleri, eleman eklenip çıkarılabilen değiştirilebilir veri yapılarıdır. Programın akışı esnasında bu veri kümesine liste elemanlarını eklemek, silmek veya listenin uzunluğunu belirlemek için yerleşik fonksiyon ve liste metotlarından yararlanılır. Bu metotların doğru kullanımı, yazılımın kararlılığını ve donanım kaynaklarının verimli tüketilmesini doğrudan etkiler.

Bir veri kümesi üzerinde işlem yapmadan önce o kümenin içinde anlık olarak kaç adet eleman bulunduğunun belirlenmesi gerekir. Python'da bir listenin toplam eleman sayısını (boyutunu) döndüren yerleşik fonksiyon, **len() (length kısaltması)** fonksiyonudur.

```
moduller = ["Menü", "Hesaplama"]
eleman_sayisi = len(moduller)
print(eleman_sayisi) # Ekrana 2 çıktısını basar.
```

Bu kod çalıştırıldığında Python'un bellek davranışı oldukça verimlidir. len() fonksiyonu listenin içindeki elemanları tek tek sayarak sisteme ekstra işlem yükü bindirmez. Python bellek mimarisinde listenin boyutu arka planda bir sayaç olarak sürekli güncel tutulur. len() çağrıldığında doğrudan bu hazır sayaç verisi okunur. Bu durum, algoritmanın çalışma hızını (zaman karmaşıklığını) optimize eden verimli bir reflekstir.

Listenin mevcut yapısını bozmadan, var olan elemanların en sonuna (kuyruğuna) yeni bir veri dâhil etmek amacıyla .append() (ekleme) metodu kullanılır.

```
sepet = [150.0, 45.5]
sepet.append(90.0)
print(sepet) # [150.0, 45.5, 90.0] çıktısını üretir.
```

.append() elemanı listenin sonuna ekler. Bu işlem genellikle verimli çalışır ancak listenin kapasitesi yetersiz olduğunda iç yapının yeniden düzenlenmesi gerekebilir.

Bir verinin listenin sonuna değil de yazılımcının belirleyeceği özel bir indeks konumuna (araya) yerleştirilmesi istendiğinde .insert() (yerleştirme) metodu devreye girer. Bu metot, .insert(hedef_indeks, eklenecek_veri) şeklinde iki parametreyle çalışır.

```
isler = ["Raporlama", "Çıkış"]
# 1. indekse "Hesaplama" verisini araya ekle
isler.insert(1, "Hesaplama")
print(isler) # ['Raporlama', 'Hesaplama', 'Çıkış'] çıktısını verir.
```

Bu kod çalıştırıldığında gerçekleşen bellek davranışı, .insert() metodunun .append() metoduna göre çok daha yüksek bir donanım maliyetine sahip olduğunu gösterir. Araya bir eleman eklendiğinde hedef konumdan sonraki elemanların indeksleri bir sağa kayar. Python yorumlayıcısı, araya yer açabilmek için sağdaki tüm verileri birer indeks sağa kaydırmak (shifting) zorunda kalır. Büyük veri kümelerinde bu metodun sıkça kullanılması yazılımın çalışma performansını düşürebilir.

Bir listenin içerisinden indeks numarasına bakmadan sadece taşımış olduğu değere göre eşleşen ilk değeri listeden kaldırmak amacıyla .remove() (silme) metodu kullanılır.

```
programlar = ["Python", "C++", "Go"]
programlar.remove("C++")
print(programlar) # ["Python", "Go"] çıktısını üretir.
```

.remove() metodu çağrıldığında Python yorumlayıcısı listeyi sıfırıncı indeksten başlayarak sağa doğru tarar (linear search). Belirtilen değere sahip ilk elemanı bulduğu an bellek hücresini boşaltır. Boşalan hücrenin yaratacağı mantıksal kopukluğu engellemek adına silinen elemanın sağ tarafında kalan tüm veriler, birer indeks sola kaydırılarak bellek bloku yeniden sıkıştırılır. Değer listede bulunmazsa ValueError oluşur.



Bilgi Kutusu

Dinamik Bellek Yönetimi ve Performans Bilinci

Programlama yaparken sadece kodun çalışması yeterli değildir. Kodun bilgisayar donanımına bindirdiği yükün de sistematik olarak hesaplanması gerekir. Listenin sonuna eleman ekleme işlemi genellikle verimlidir, listenin başına .insert(0, ...) ile ekleme yapmak ise mevcut elemanların indekslerinin kaydırılmasını gerektirir. Geliştirdiğiniz algoritmalarda doğru metodu doğru ihtiyaç için seçmek, sistem kaynaklarını koruyan vizyoner ve sistematik bir yazılımcı eğilimidir.

3.2.11. Döngü ve Liste İşlemlerinde Yapay Zekâ Denetimi

Büyük dil modellerine (LLM) dayalı yapay zekâ asistanları; karmaşık döngüleri, koşul bloklarını ve liste metotlarını saniyeler içinde yazarak yazılımcılara büyük bir hız kazandırabilir. Bu asistanların çalışma mantığı, kodun doğruluğunu analitik olarak **anlamaya** değil geçmiş verilerden öğrendikleri dil kalıplarına göre en olası bir sonraki **kelimeyi / karakteri tahmin etmeye** dayanır. Yapay zekâ modellerinin koşul, döngü ve liste kurgularında en sık yaptığı üç kronik hata türü şunlardır:

Birinci kronik hata, gizli yapısal kusur olan kararsız girinti hatalarıdır. Python dil mimarisinde kod bloklarının sınırları, süslü parantezlerle değil sadece boşluk düzeyleriyle (girintilerle) belirlenir. Yapay zekâ modelleri, özellikle iç içe koşul yapısı veya döngü içeren uzun kod blokları üretirken girinti hizalamasını kaybedebilir. Bu noktada yapay zekânın en sık düştüğü hata; gözle bakıldığında aynı hizada duran bazı satırlarda 4 adet **boşluk** karakteri, bazı satırlarda ise 1 adet **Tab** karakteri kullanmasıdır. Bu durum program çalıştırıldığı an IndentationError veya TabError fırlatılmasına yol açar. Bu gizli kusura karşı uygulanacak en güvenli insan denetimi çözümü; yapay zekâdan alınan kodları doğrudan kopyalamak yerine editörün **Render Whitespace (Görünmez Boşlukları Göster)** özelliğini açarak, noktaları ve okları sistematik olarak kontrol etmektir.

İkinci kronik hata, sonsuz döngü tuzağıdır. Yapay zekâ modelleri, koşula bağlı olarak dönen while döngülerini kurarken döngüyü sonlandıracak mantıksal anahtarı (break ifadesini veya sayaç güncellemesini) döngü blokunun içine yerleştirmeyi unutabilir veya yanlış koşul basamağına yazabilir. Bu hata gözden kaçtığında ve kod çalıştırıldığında sayac değeri döngü içinde hep 1 kalacağı için uygulamanın yanıt vermemesine veya gereksiz işlemci kullanmasına yol açabilir. Analitik bir programcının bu tuzağa karşı uygulayacağı insan denetimi çözümü; döngü koşulunu etkileyen değişkenin, döngünün içinde ve doğru girinti seviyesinde güncellendiğini manuel olarak doğrulamaktır.

Üçüncü kronik hata ise indeks sınır taşması ve metot sapmalarıdır. Yapay zekâ araçları, listeler üzerinde for döngüsü kurarken veya listeden .remove() metoduyla eleman silerken sınır değerlerini sıklıkla karıştırır. Yapay zekânın buradaki en yaygın hatası, Python listelerinin 0 (sıfır) tabanlı indeksleme mimarisine sahip olduğunu unutarak listelerin boyutuyla indeks numaralarını eşitlemesidir. Örneğin 5 elemanlı bir listenin son elemanına ulaşmak için liste[5] kodunu üretebilir. Bu durum doğrudan IndexError: list index out of range hatasına yol açar. Bilişim uzmanlarının bu sapmalara karşı alacağı insan denetimi çözümü ise listenin sınırlarını ve metot çağırma işlemlerini yürütmeden önce len() fonksiyonu ile listenin anlık boyutunu denetlemek ve program akışına sınır değer kontrolü (if eleman in liste:) yerleştirmektir.

Bilişim dünyasında yapay zekâ modelleri; çok hızlı yazabilen, her şeye anında onay veren ama son derece aceleci, dikkatsiz bir stajyer gibi konumlandırılabilir. Hazır bir kod satırının bilgisayarda çalıştırılmadan önce girdilerinin, döngü kırılma noktalarının ve indeks sınırlarının zihinsel süzgeçten geçirilmesi mesleki bir sorumluluktur. Yapay zekânın ürettiği bir koddaki mantık hatasının çalıştırılmadan önce tahmin edilebilmesi, yazılımcılara Yapay Zekâ Denetçisi unvanını kazandırır.

6. UYGULAMA

Uygulamanın Adı: HATA AVCILIĞI VE YAPAY ZEKÂ KOD DENETİMİ

Uygulamanın Süresi: 30 dakika

Uygulamanın Amacı: Yapay zekâ tarafından üretilmiş hatalı kodları çalıştırmadan önce analiz etmek; girinti, döngü ve indeks hatalarını belirleyip düzelterek kod kararlılığını sağlamak

Problem Senaryosu: Bir yapay zekâ asistanına "Kullanıcının sepetindeki fiyatları listeye ekleyen, ÇIKIŞ yazana kadar dönen ve sonunda sepeti listeleyen bir menü yaz." komutu verilmiştir. Yapay zekânın ürettiği kodlar çalıştırılmadan önce insan denetiminden geçirilmesi istenmektedir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel şekilde gerçekleştiriniz.

Uygulamanın Aşamaları

Kod Dosyasının Hazırlanması ve İncelenmesi

a) Yapay zekâ, bu kodu üretirken (olasılıksal doğası gereği) kodun içine 3 adet kronik hata yapmıştır. Kod bloklarını bilgisayarda **çalıştırmadan önce** satır satır inceleyiniz. Tespit ettiğiniz hataların yerini, türünü ve programın kararlılığını nasıl etkileyeceğini **Hata Tespit Masası** tablosuna kaydediniz.

b) Kod blokunu kod editörüne aktararak programın hatasız, kararlı ve PEP 8 stil önerilerine uygun temiz sürümünü yeniden yapılandırınız.

Hata Tespit Çalışması Tablosu

Hata Türü	Hatayı Barındıran Satır Aralığı	Kök Neden ve Mantıksal Kusur
Yapısal Hata (IndentationError)		
Mantıksal Sınır Hatası (IndexError)		

6. UYGULAMA

```
# ============================================================
# YAPAY ZEKÂ TARAFINDAN ÜRETİLEN KARARSIZ KOD TASLAĞI
# DİKKAT: Bu kod mantıksal ve yapısal hatalar barındırmaktadır!
# ============================================================

sepet_listesi = [100.0, 250.0, 75.0]
print("--- Yapay Zekâ Sepet Yönetim Sistemi ---")

while True:
    girdi = input("Ürün fiyatı giriniz veya çıkmak için 'ÇIKIŞ' yazınız: ")

    if girdi.upper() == "ÇIKIŞ":
        print("Hesaplama yapılıyor...")
        break

    else:
        fiyat = float(girdi)
        sepet_listesi.append(fiyat)
        print(f"Ürün eklendi. Sepetteki Ürün Sayısı: {len(sepet_listesi)}")

sayac = 0
while sayac < 3:
    print(f"Doğrulanıyor... İşlem Sırası: {sayac}")
sayac += 1

son_indeks = len(sepet_listesi)
print(f"En son eklenen ürünün fiyatı: {sepet_listesi[son_indeks]} TL")
```

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

3.3. GRAFİKSEL GÖRSELLEŞTİRME

Programlama dillerinde verileri işlemeyi, koşulları yönetmeyi ve döngüler kurmayı öğrenmek, algoritmik düşünme becerisinin temelini oluşturur. Yazılan kodların sadece terminal ekranında metinsel çıktılar (sayılar, yazılar vb.) üretmesi, programın arka planda yürüttüğü geometrik ve mantıksal dönüşümlerin görsel olarak algılanmasını zorlaştırabilir. Bir mimarın hesaplamalarla dolu kâğıt üzerindeki projesinin, üç boyutlu bir makete veya gerçek bir binaya dönüşmeden zihinde tam olarak canlandırılamaması gibi bilgisayarın belleğindeki sayısal veriler de görselleştirilmeden tam olarak kavranamayabilir. Yazılım dünyasında soyut algoritmaları çizgilere, renklere ve piksellere dönüştürerek somutlaştırma işlemine **grafiksel görselleştirme (graphical visualization)** adı verilir. Python dil mimarisi, kodların ekranda görsel birer sanat eserine veya grafik arayüzüne dönüşmesini sağlayan güçlü kütüphaneler barındırır.

3.3.1. Koordinat Tabanlı Çizim Aracı: Turtle

Python yerleşik kütüphanesinde bulunan turtle (kaplumbağa), kodlarla iki boyutlu düzlemde çizim yapmayı sağlayan eğitsel bir grafik aracıdır. Bu modülde ekranda görünmez bir kartezyen koordinat düzlemi (X ve Y eksenleri) ve bu düzlem üzerinde hareket eden, elinde bir boyama kalemi tutan sanal bir imleç bulunur.

Günlük hayatta gözleri kapalı birine elinde bir fırçayla geniş ve boş bir tuval üzerinde yürüyerek, resim çizdirilmek istendiğinde ona **on adım ileri git, doksan derece sağa dön, fırçanın rengini kırmızı yap** gibi çok net ve sıralı yönlendirmeler verilmesi gerekir. Turtle modülü de bu yönlendirme mantığıyla çalışır. Sanal imlecin baktığı yön, dönüş açısı, kalem rengi gibi parametreler döngülerle birleştirildiğinde bilgisayar ekranında karmaşık geometrik desenler saniyeler içinde oluşturulabilir. Bu süreç, öğrencilerin algoritmik talimatları piksellere dönüştürerek sonuçlarını somut bir şekilde değerlendirmesini ve soyut matematiksel açıları görsel birer çıktıya dönüştürmesini sağlar. Vektörel çizim süreçlerinde algoritmik desenler oluşturmak ve imleci (turtle) çalışma ekranında yönlendirmek amacıyla sıklıkla kullanılan temel grafik komutları ile bu komutların işlevleri Tablo 3.2'de referans olarak özetlenmiştir.

Tablo 3.2: Temel Turtle Komutları ve İşlevleri Referans Rehberi

Komut	Kısa Yazım	İşlevi ve Davranışı
forward(mesafe)	fd(mesafe)	İmleci baktığı yöne doğru belirtilen piksel kadar ileri taşır.
right(açı)	rt(açı)	İmlecin yönünü saat yönünde belirtilen derece kadar döndürür.
left(açı)	lt(açı)	İmlecin yönünü saat yönünün tersine belirtilen derece kadar döndürür.
pencolor(renk)	-	Çizgi rengini belirler (örneğin "blue", "red").

8. ÖRNEK

DÖNGÜYLE GEOMETRİK ŞEKİL MODELLEME (KARE ÇİZİMİ)

Uygulama Süresi: 20 Dakika

Yazılımın görselleştirme modülünde, ekranda bir kare çizilmesi hedeflenmektedir. Bir kare çizmek için program akışının ardışık olarak **100 piksel ileri git ve 90 derece sağa dön**. komutunu tam 4 kez tekrarlaması gerekir. Bu eylemin alt alta 4 kez (toplam 8 satır) kopyalanması yerine for döngüsü kullanılarak sürecin nasıl kısaltılacağı ve soyut mantığın piksellere nasıl modelleneceği test edilecektir.

Örnek Kod Bloku

```
import turtle

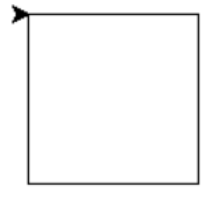
# Grafik penceresini ve çizim kalemini hazırla.
kalem = turtle.Turtle()

# VERİMLİ DÖNGÜSEL MODELLEME: Aynı 2 satırı kopyalamak yerine 4 kez döndür.
for sayac in range(4):
    kalem.forward(100) # 100 piksel ileri git.
    kalem.right(90)    # 90 derece sağa dön (kare köşesi).

turtle.done() # Çizim bittiğinde pencereyi açık tut.
```

Kod Analizi

1. **Döngü Başlatma**: Python yorumlayıcısı range(4) fonksiyonunu gördüğünde 0, 1, 2 ve 3 değerlerini sırayla sağlayan bir range nesnesi oluşturur.
2. **Görsel İşleme**: Döngünün her bir turunda imleç (kaplumbağa) 100 piksel boyunca düz bir çizgi çizer ve bulunduğu yöne göre saat yönünde 90 derece döner.
3. **Karenin Tamamlanması**: Bu eylem tam 4 kez kusursuzca tekrarlandığında imleç başlangıç noktasına geri döner ve ekranda kapalı bir kare şekli oluşur (Görsel 3.1). turtle.done() komutu sayesinde grafik penceresi kapanmadan ekranda kararlı bir şekilde tutulur.



Görsel 3.1: Kare uygulamasının çıktısı

7. UYGULAMA

Uygulamanın Adı: DÖNGÜSEL GEOMETRİ VE GRAFİKSEL TASARIM

Uygulamanın Süresi: 30 dakika

Uygulamanın Amacı: for döngüsü, range() fonksiyonu ve turtle kütüphanesini kullanarak çokgen desenleri oluşturmak, açı değişimlerinin görsel çıktıya etkisini analiz etmek

Çokgen desenleri oluşturmak amacıyla aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Kod Dosyasının Hazırlanması ve İncelenmesi

a) Kod editöründe kod_asistan_grafik.py isimli yeni bir dosya oluşturunuz.

b) import turtle satırıyla grafik kütüphanesini projenize dâhil ediniz.

c) Kullanıcıdan bir döngü sınırı belirlemek üzere kaçgen çizmek istediğini (kenar_sayisi) dinamik girdi olarak alınız.

ç) Geometrik kural gereği, bir düzgün çokgenin dış açılar toplamı 360 derecedir. Döngü içinde kullanılacak dönme açısını aci = 360 / kenar_sayisi formülüyle programa hesaplatınız.

d) Hesaplanan bu verileri for döngüsüne bağlayarak dinamik çokgen çizimini kaydediniz.

```
# KOD-ASİSTAN PROJESİ v0.3 - DÖNGÜSEL GRAFİK VE GEOMETRİ MODÜLÜ
# Geliştirici: [Öğrenci Adı Soyadı]
# Standart: PEP 8 / Disiplinler Arası Matematik ve Döngü Entegrasyonu
import turtle
print("=== KOD-ASİSTAN: DÖNGÜSEL GRAFİKSEL GÖRSELLEŞTİRME ===")
# Kullanıcıdan dinamik geometrik veri girişi alınması
kenar_sayisi = int(input("Çizilmesini istediğiniz çokgenin kenar sayısını giriniz (Örn: 3, 5, 6, 8):
"))
if kenar_sayisi < 3:
    print("Hata: Bir çokgen oluşturabilmek için en az 3 kenara ihtiyaç vardır!")
else:
    # MATEMATİKSEL MUHAKEME ALANI
    # Geometrik dış açı formülü Python yorumlayıcısına hesaplatılır.
    donme_acisi = 360 / kenar_sayisi
    print(f"\nHesaplanan Dış Açı: {donme_acisi:.1f} Derece.")
    print("Grafik motoru tetikleniyor, lütfen açılan pencereyi izleyiniz...")
    # Çizim kalemi ayarları
    asistan_kalem = turtle.Turtle()
```

7. UYGULAMA

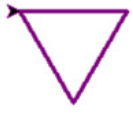
```
asistan_kalem.pencolor("purple") # Çizgi rengi
asistan_kalem.pensize(3)       # Çizgi kalınlığı
# DİNAMİK GRAFİK DÖNGÜSÜ
# Döngü, kullanıcının girdiği kenar sayısı kadar kusursuzca döner.
for sira in range(kenar_sayisi):
    asistan_kalem.forward(80)          # Her bir kenarın uzunluğu 80 piksel
    asistan_kalem.right(donme_acisi)  # Hesaplanan dinamik açı kadar dönüş

print("Görselleştirme algoritması başarıyla tamamlandı.")
turtle.done()
```

2. Aşama: Etkileşimli Deney

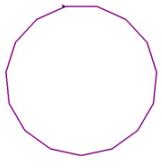
Yazdığınız kodu terminalde çalıştırarak şu değerleri giriniz ve döngülerin sanatsal gücünü analiz ediniz:

a) Deney (Üçgen Kararlılığı): Kenar sayısına 3 giriniz. Python yorumlayıcısının 120 derece dönerek bir eşkenar üçgen oluşturmasını izleyiniz (Görsel 3.2).



Görsel 3.2: Üçgen çıktısı

b) Deney (Çember Algısı): Kenar sayısına 15 giriniz. Çizim hızlandıkça ekranda oluşan şeklin neden bir çembere benzediğini, kenar sayısı arttıkça piksellerin mantıksal dağılımını arkadaşlarınızla tartışınız (Görsel 3.3).



Görsel 3.3: On beş kenarlı çokgen çıktısı

c) Deney (Spiral Tasarımı): Döngünün içindeki asistan_kalem.forward(80) satırını döngü değişkenine bağlı olarak asistan_kalem.forward(sira * 2) şeklinde güncelleyip kenar sayısını 20 giriniz. Ekranda oluşan spiral yapıyı döngü sayaçlarının değişimi üzerinden yorumlayınız (Görsel 3.4).



Görsel 3.4: Spiral kod çıktısı

7. UYGULAMA

3. Aşama: Projenin Dijital Depoya Yüklenmesi

a) Proje dosyanızı kaydediniz.

b) Dijital depo hesabınızdaki kod-asistan-v0.2 deponuza giderek **Add file > Upload files** adımlarıyla dosyanızı deponuza yükleyiniz.

c) Sürüm notu alanına kurumsal şeffaflık standartlarına uygun olarak şu notu ekleyiniz: "turtle kütüphanesiyle döngüsel grafik görselleştirme Kod-Asistan projesine eklendi (v0.3)."

Değerlendirme: Çalışmanızı, karekodda yer alan Öz Değerlendirme Formu'ndaki ölçütlere göre değerlendiriniz. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Kendi çalışmanızı değerlendirmek için karekodda yer alan Öz Değerlendirme Formu'nu doldurunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

3.3.2. İç İçe Döngülerle Karmaşık Geometrik Desenlerin Üretimi

İç içe döngülerin çalışma mekanizması dış döngü ve iç döngü hiyerarşisine dayanır. Dışarıdaki döngü sadece 1 tur döndüğünde içerideki döngü başlangıçtan bitişe kadar olan tüm turlarını tamamlamak zorundadır. İç döngü tamamen bitmeden dış döngü bir sonraki adımına geçemez.

Günlük hayatta bir analog saatin çalışma mekanizması, iç içe döngülerin bu hiyerarşik yapısına verilebilecek somut örneklerden biridir. Bir saatte yer alan akrep dış döngüyü, yelkovan ise iç döngüyü temsil eder. Akrebin yalnızca bir sayılık ilerleme kaydedebilmesi (dış döngünün bir tur atması) için yelkovanın 60 dakikalık tam bir turu eksiksiz bir şekilde bitirmesi şarttır. Yelkovan kendi turlarını tamamlamadan akrep kesinlikle bir sonraki saate ilerleyemez. Bu mekanizmada olduğu gibi programlama dillerinde de iç döngünün tüm tekrarları bütünüyle sonlanmadan dış döngünün bir sonraki adıma geçmesine izin verilmez. Bu mutlak bağımlılık, bilgisayarın karmaşık işlemleri sırayla, düzenli ve hatasız bir şekilde yapılandırmasını mümkün kılar. Bu mekanizma; turtle grafik kütüphanesinin yön, piksel ve açı parametreleriyle birleştiğinde iç döngünün temel bir geometrik şekli (örneğin bir üçgen veya kareyi) çizdiği, dış döngünün ise bu şekli her seferinde belirli bir açıyla döndürerek ekranda muazzam simetrik geometrik desenler ve geometrik spiraller oluşturduğu akıllı bir tasarım sistemine dönüşür.

Geleneksel mimaride ve çini sanatında ustaların karmaşık eserlerini üretme süreci, bu ardışık çizim mimarisinin ekrandaki yansımasına verilebilecek en estetik örneklerden biridir. Bir usta, önce merkezdeki tek bir geometrik motifi tüm detaylarıyla çizer. Bu adım, yazılımdaki iç döngüyü temsil eder. Ardından elde edilen bu temel motif, belirli bir açıyla kaydırılarak zemin etrafında defalarca kopyalanır. Bu aşama ise dış döngünün kontrolündedir. Sadece basit bir çizimin iç içe döngüler aracılığıyla aynı eksen etrafında döndürülerek çoğaltılması, matematiksel hesaplamaların görsel bir derinlik kazanmasını sağlar. Yüzlerce satır tekrarlı komut dizisi yazmak yerine sadece birkaç satırlık iç içe yapı kurgulamak, sistemin doğadaki kusursuz simetriye benzeyen fraktalları hızlı ve hatasız şekilde inşa etmesini sağlar.

8. UYGULAMA

Uygulamanın Adı: İÇ İÇE DÖNGÜLERLE DÖNDÜRÜLMÜŞ ÜÇGEN DESENİ

Uygulamanın Süresi: 30 dakika

Uygulamanın Amacı: İç içe for döngülerini kullanarak turtle ile geometrik şekiller üretmek, iç ve dış döngülerin birlikte nasıl karmaşık desenler oluşturduğunu kavramak

Geometrik desenler oluşturmak amacıyla aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Kod Dosyasının Hazırlanması ve İncelenmesi

a) Kod editöründe kod_asistanim çalışma klasörünüzün içine kod_asistan_fraktal.py isimli yeni bir dosya oluşturunuz.

b) import turtle satırıyla grafik kütüphanesini projenize dâhil ediniz.

c) Ekrana 36 adet üçgen çizebilmek için dış döngüyü 36 kez çalışacak (for dis_sira in range(36):) şekilde kurgulayınız.

ç) Dış döngünün içine, bir eşkenar üçgenin çizim adımlarını yönetecek 3 turluk bir iç döngü (for ic_sira in range(3):) yerleştiriniz.

d) İç döngünün her üçgen çizimini bitirmesinin ardından, şekillerin üst üste binmemesi için dış döngüye asistan_kalem.right(10) komutunu ekleyerek imleci her adımda 10 derece döndürünüz.

```
# Geliştirici: [Öğrenci Adı Soyadı]
# Standart: PEP 8 / İç İçe for Döngüleri ve Grafiksel Matris Yönetimi
import turtle
print("=== KOD-ASİSTAN: AKTİF DÖNDÜRÜLMÜŞ ÜÇGEN DESENİ ===")
print("Grafik penceresi açılıyor, iç içe döngülerin mekanizmasını izleyiniz...\n")
# Çizim aracının hazırlanması ve kararlı grafik ayarları
asistan_kalem = turtle.Turtle()
asistan_kalem.speed(0)      # Çizimi hızlandırmak için en yüksek hız (0) modu
asistan_kalem.pencolor("blue") # Mavi tasarım rengi
asistan_kalem.pensize(2)
# ============================================================
# MANTIKSAL HİYERARŞİ: İÇ İÇE DÖNGÜ MOTORU
# ============================================================
# 1. DIŞ DÖNGÜ: Toplam şekil adedini ve dönüş eksenini yönetir.
# Bu döngü 36 kez dönecek, yani ekranda 36 adet geometrik şekil oluşacaktır.
for dis_sira in range(36):
    # 2. İÇ DÖNGÜ: Temel geometrik şekli (Eşkenar Üçgen) çizer.
  # Dış döngünün HER BİR TURUNDA bu iç döngü 3 turunu tamamen bitirir.
  for ic_sira in range(3):
    asistan_kalem.forward(120)  # 120 piksel ileri git
    asistan_kalem.left(120)     # Eşkenar üçgen dış açısı (120 derece) sola dön
```

8. UYGULAMA

```
        # İÇ DÖNGÜNÜN BİTİŞ ALANI
    # Bir adet üçgen çizimi bitti! Şimdi dış döngünün kontrolündeyiz.
    # Bir sonraki üçgenin bir öncekinin üstüne binmemesi için ekseni 10 derece kaydırıyoruz.
    asistan_kalem.right(10)
print("Algoritma Tamamlandı: 36 adet üçgen, iç içe döngü mimarisiyle işlendi.")
turtle.done()
```

2. Aşama: Etkileşimli Deney

Yazdığınız aktif kodlama örneğini terminalde yürüterek şu maddeleri analitik süzgecinizden geçiriniz:

a) Deney (Python Yorumlayıcısı Tur Muhakemesi): Bu program tamamen durduğunda iç döngünün en içindeki asistan_kalem.forward(120) satırı program akışı boyunca toplam kaç kez çalıştırılmıştır? Matematiksel olarak gerekçelendiriniz.

b) Deney (Geometrik Değişim): İç döngüdeki range(3) ifadesini range(4) yapıp, içindeki açıyı asistan_kalem.left(90) olarak güncelleyiniz. Programı yeniden çalıştırdığınızda oluşan yeni tasarımın temel geometrik kökenini analiz ediniz.

c) Deney (Renk Kararlılığı): İç döngünün hemen dışına, dış döngünün ise içine şu iki satırı ekleyerek her turda renk geçişini simüle ediniz. Döngü değişkenlerinin dinamik yapısını görsel olarak yorumlayınız.

```
if dis_sira % 2 == 0:
    asistan_kalem.pencolor("purple")
else:
    asistan_kalem.pencolor("blue")
```

3. Aşama: Projenin Dijital Depoya Yüklenmesi

a) İç içe döngü algoritmasıyla kararlı ve estetik bir görselleştirme katmanı oluşturduğunuz kod_asistan_fraktal.py kaynak kod dosyanızı kaydediniz.

b) Dijital depo hesabınızdaki kod-asistan-v0.2 deponuza giderek **Add file > Upload files** adımlarıyla dosyanızı deponuza yükleyiniz.

c) Sürüm notu alanına şu kurumsal açıklamayı kaydediniz: "Grafik modülüne iç içe for döngüleri (nested loops) mimarisi kullanılarak 36 kez döndürülmüş üçgen deseni entegre edildi (v0.3)."

Değerlendirme: Çalışmanızı, karekodda yer alan Öz Değerlendirme Formu'ndaki ölçütlere göre değerlendiriniz. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Kendi çalışmanızı değerlendirmek için karekodda yer alan Öz Değerlendirme Formu'nu doldurunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

3.3.3. Dijital Depolarda Sürüm Yönetimi ve Release Kavramı

Bir yazılım projesi geliştirilirken kodlar sürekli olarak güncellenir, yeni özellikler eklenir ve hatalar ayıklanır. Yazılımın belirli aşamalarda ulaştığı **güvenli, test edilmiş** ve **kararlı (stabil)** aşamaları dondurarak kayıt altına almak gerekir. Yazılım sektöründe projenin kullanıcıya sunulmaya hazır, kararlı her bir ana basamağına **sürüm (version)** adı verilir.

Sürüm numaralandırmaları genellikle **v1.0, v2.0** şeklinde standart bir mimariyle yapılır. Burada yer alan v1.0, projenin ilk tamamlanmış, tüm temel fonksiyonları kararlı çalışan resmî **final sürümü** anlamına gelir.

Dijital depolama ve sürüm kontrol platformu üzerinde projenin belirli bir andaki hatasız kopyasını etiketleyerek indirilebilir bir paket hâlinde dünyaya ilan etme işlemine **Release (Yayın / Sürüm Oluşturma)** denir. Release oluşturulduğunda sistem, kaynak kodları otomatik şekilde .zip formatında sıkıştırarak arşivler ve projenin o anki sürümünü koruma altına alır.

9. UYGULAMA

Uygulamanın Adı: İLK RESMÎ SÜRÜM (v1.0) VE RELEASE YÖNETİMİ

Uygulamanın Süresi: 30 dakika

Uygulamanın Amacı: Kod-Asistan projesinde koşullar, döngüler, liste metotları ve görselleştirme modüllerini birleştirerek projenin ilk kararlı sürümünü dijital depoda yayımlama becerisi kazanmak

Kod-Asistan projesini dijital depoda yayımlamak amacıyla aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Projenin Dijital Depoya Yüklenmesi

a) İnternet tarayıcınızdan dijital depo hesabınıza giriş yapınız ve kod-asistan isimli proje deponuzu açınız.

b) Deponun ana sayfasının sağ tarafında yer alan **Releases (Yayınlar)** başlığı altındaki **Create a new release (Yeni bir yayın oluştur)** düğmesine tıklayınız.

c) **Choose a tag (Bir etiket seçin)** açılır menüsüne tıklayarak, projenizin ilk resmî sürüm numarası olan v1.0 ifadesini yazıp **Create new tag** seçeneğine basınız.

ç) **Release title (Yayın başlığı)** alanına kurumsal standartlara uygun olarak Kod-Asistan v1.0 - İlk Kararlı Sürüm (Final) başlığını atınız.

9. UYGULAMA

d) **Describe this release (Bu yayını açıkla)** alanına projenizin neleri başardığını gösteren sürüm notlarını (markdown biçiminde) şu şekilde ekleyiniz:

```
## Kod-Asistan v1.0 Sürüm Notları
Bu sürüm, projenin ilk tamamlanmış resmî final sürümüdür.

### Eklenen Modüller ve Özellikler:
* **Koşul Yapısı:** Kullanıcı taleplerini süzebilen dinamik karar blokları eklendi.
* **Döngü Kararlılığı:** `while` ve `for` motorlarıyla hata avcılığı denetimleri yapıldı.
* **Görselleştirme:** `turtle` entegrasyonuyla iç içe döngülü grafik katmanı eklendi.
```

e) Sayfanın en altında yer alan yeşil renkli **Publish release (Yayınla)** düğmesine tıklayarak sürecinizi tamamlayınız.

2. Aşama: Sürümünü Savunma ve Akran Değerlendirmesi

Projenizin v1.0 sürümünü başarıyla yayınladıktan sonra şu analitik adımları takip ediniz:

a) **Analiz (Arşiv Denetimi):** Release işlemi bittikten sonra ekranınıza gelen sayfadaki Source code (zip) bağlantısına tıklayarak projenizi bilgisayarınıza indiriniz. Sıkıştırılmış dosya içindeki kaynak kodların, bilgisayarınızdaki kodlarla uyumlu olduğunu doğrulayınız. Sürüm kontrolünün projenizi bilgisayar çökmelerine karşı nasıl koruduğunu yorumlayınız.

b) **Akran Savunması:** Sınıftan sıra arkadaşınızla eşleşiniz. Arkadaşınızın GitHub deposuna giderek onun oluşturduğu v1.0 sürüm notlarını okuyunuz. Projenizin README dosyasına ve kaynak kod kalitesine (değişken isimlendirmeleri, yorum satırları vb.) bakarak birbirinize yapıcı geri bildirimler veriniz.

Değerlendirme: Çalışmanızı, karekodda yer alan Öz Değerlendirme Formu'ndaki ölçütlere göre değerlendiriniz. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Kendi çalışmanızı değerlendirmek için karekodda yer alan Öz Değerlendirme Formu'nu doldurunuz.

Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

10. UYGULAMA

Uygulamanın Adı: SONSUZ DÖNGÜ KAPANI VE GEOMETRİK FRAKTAL SANATI

Uygulama Süresi: 30 Dakika

Problem Tanımı: Kod-Asistan projenizin görsel vitrini için bir dijital sanat motoru tasarlamanız istenmektedir. Sistem, kullanıcıdan kaç eksenli bir fraktal çizmek istediğini interaktif olarak alacak ve turtle grafik ekranında çizime başlayacaktır.

Çizim algoritması teorik olarak sonsuz bir döngüye (while True) sahip olacaktır. Programın sonsuz döngüde kalmasını engellemek için; çizgi uzunluğu 300 piksele ulaştığı an sistemi break komutuyla kırmalı ve çizimi güvenli bir şekilde sonlandırmalıdır.

Uygulama Adımları

a) Kod editöründe kod_asistanim çalışma klasörünüzün içine fraktal_sanati.py adında temiz bir Python kaynak kod dosyası açınız.

b) Programın en üstüne Python'ın standart grafik motoru olan turtle kütüphanesini dâhil ediniz.

c) Kullanıcıdan döngüsel dönüş açısını belirlemek üzere dinamik bir sayısal veri talep ediniz (Matematiksel ipucu: İç içe tam döngü için 360 / eksen_sayisi formülünü kullanabilirsiniz.).

ç) Çizgilerin her adımda büyümesini sağlayacak bir cizgi_boyu = 0 sayacı tanımlayınız.

d) Bir while True: ile döngü bloku başlatınız.

e) Döngünün içinde turtle.forward(cizgi_boyu) ve turtle.left(aci) komutlarıyla kaplumbağayı hareket ettiriniz. Her turun sonunda cizgi_boyu değişkenini 2 piksel artırarak spiral etkiyi yakalayınız.

f) Döngünün tam ortasına bir if koşulu yerleştiriniz. Eğer cizgi_boyu >= 300 olduysa ekrana f-string ile "Program Güvenliği Sağlandı: 300 piksel sınırında sonsuz döngü kırıldı." yazdırıp break komutuyla döngü kapanından kaçınız.

g) turtle.done() komutuyla çizim bittiğinde ekranın açık kalmasını sağlayınız.

ğ) Kodunuzu hatasızlaştırdıktan sonra dijital deponuza "Turtle tabanlı spiral çiziminde sınır kontrollü döngü ve break kullanımı eklendi." notuyla gönderiniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

2. SIRA SİZDE

1. Kontrol Yapıları ve Döngüler öğrenme birimi boyunca gerçekleştirdiğiniz tüm uygulamaları, tasarım süreçlerinizi ve Kod-Asistan çalışmalarınızı gözünüzün önünden geçiriniz.
2. Aşağıda üç farklı geometrik sembolle tanımlanan alanlardaki (Üçgen, Kare ve Çember) yönlendirme sorularını dikkatlice okuyunuz.
3. Öğrenme birimi konularıyla ilgili öğrendiklerinizden hareketle kendi öğrenme yolculuğunuzu yansıtan içten ve özgün cevaplarınızı verilen boşluklara yazınız.

ÜÇGEN: Zihnimde Şekillenen 3 Temel Kazanım

Bu öğrenme biriminin sonunda artık tam anlamıyla uzmanlaştığımı düşündüğüm, başkasına hiç zorlanmadan anlatabileceğim **3 temel programlama becerisi** şunlardır:

1. .....
2. .....
3. .....

KARE: Zihnimi "Köşeye Sıkıştıran" ve Netleşmesi Gereken Pürüzler

Bu öğrenme biriminde beni hâlâ zorlayan, zihnimde tam olarak oturmayan ve **öğretmenime sormam gereken köşeli / pürüzlü noktalar** şunlardır:

- .....

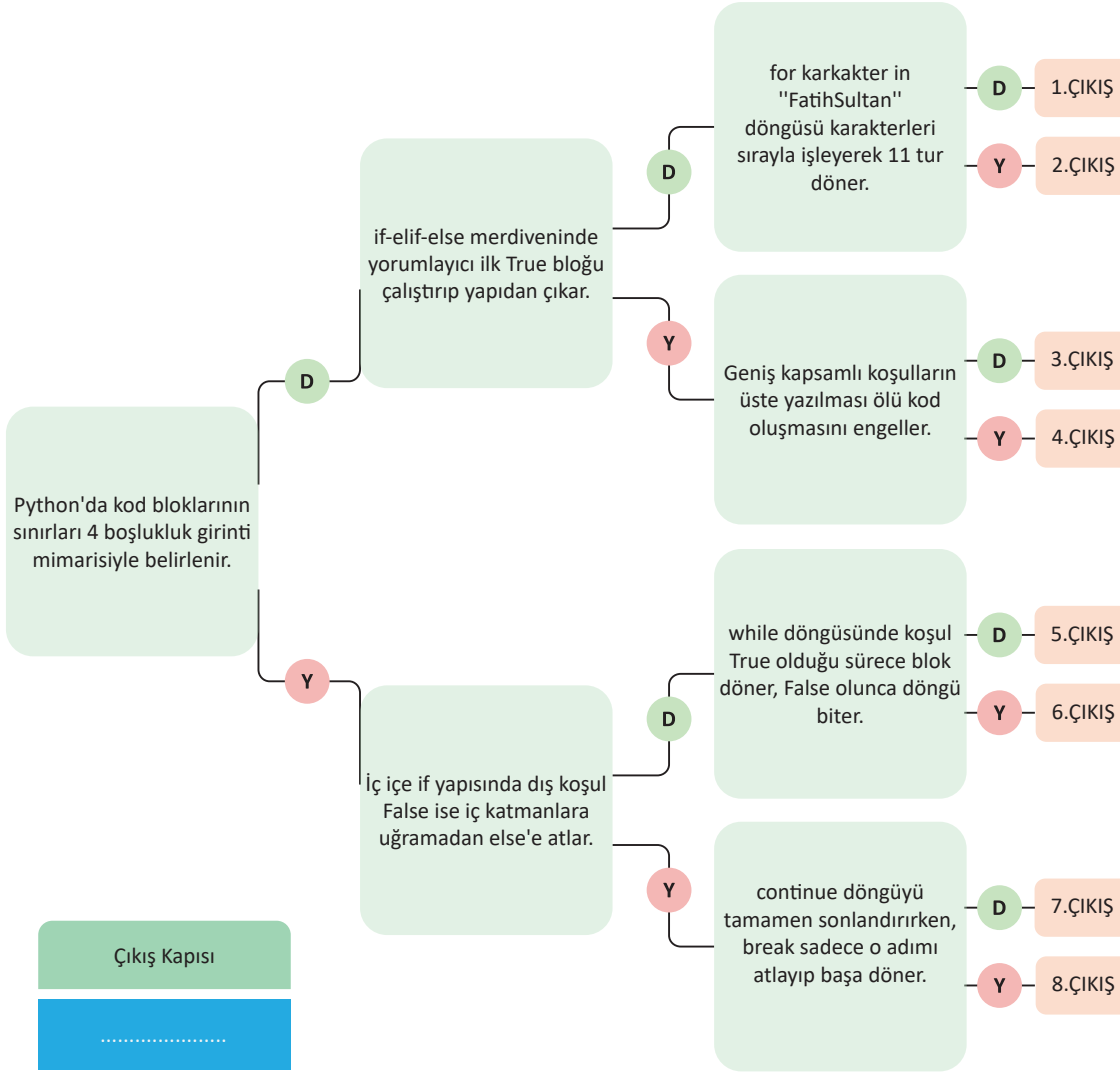
ÇEMBER: Zihnimde Dönen ve Hayatıma Değer Katan Etik Boyut

Yazılım projelerinde karmaşık döngüleri ve listeleri kurgularken yapay zekânın yapabileceği kronik hatalara karşı İnsan Denetimi prensibini işletmenin önemi ve hazırlanan yazılımı dijital depolarda dünyaya sunarken (Release) kurumsal şeffaflık ve dürüstlük ilkelerine uygun sürüm notları yazmanın mesleki etiğe katkısı hakkında **zihnimde yer eden ve hayatım boyunca dikkat edeceğim temel düşünce** şudur:

- .....

ÖLÇME VE DEĞERLENDİRME

A) Aşağıdaki Tanılayıcı Dallanmış Ağaç şemasında yer alan 1. ifadeden (kökten) başlayarak okuma yapınız. Okuduğunuz ifadenin doğru (D) veya yanlış (Y) olduğuna karar vererek ilgili ok yönünde (dallanarak) bir sonraki ifadeye ilerleyiniz. Her yol ayrımında verdiğiniz kararlar sizi en son aşamada 8 farklı çıkış kapısından yalnızca birine ulaştıracaktır. Kendi zihinsel mantık akışınızı takip ederek ulaştığınız Çıkış Kapısı Numarasını sayfa altındaki kutucuğa işaretleyiniz.



ÖLÇME VE DEĞERLENDİRME

B) Aşağıda 1'den 12'ye kadar numaralandırılmış kutucuklardan oluşan yapılandırılmış grid tablosunu ve altındaki soruları dikkatlice inceleyiniz. Her sorunun cevabı veya cevapları tablodaki kutucuklarda yer almaktadır. Bir kutucuğu birden fazla sorunun cevabı olarak kullanabilirsiniz. Soruların altındaki boşluklara, doğru olduğunu düşündüğünüz kutucuk numaralarını yazınız.

if-elif-else (1)	break (2)	**.lower(). (3)**
while (4)	.append(). (5)	**IndentationError (6)**
continue (7)	.insert() (8)	**for (9)**
else (10)	Release (11)	**İç İçe Döngü (12)**

a) Döngüyü doğrudan oluşturan veya iç içe döngü yapısını temsil eden kutucuklar hangileridir?
.....

b) Döngü akışını erken sonlandıran veya bir turu atlayan ifadeler hangileridir?
.....

c) Listeye eleman ekleyen metotlar hangileridir?
.....

ç) Karar yapısında kullanılan anahtar kelime veya yapılar hangileridir?
.....

d) Python'da blok yapısı, girinti veya hata yönetimiyle ilişkilendirilebilecek kutucuklar hangileridir?
.....

e) Kullanıcı etkileşimli bir menü programında birlikte kullanılabilecek kutucuklar hangileridir?
.....

ÖLÇME VE DEĞERLENDİRME

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1.

```
yas = 17
toplam_sepet = 600
kupon_aktif = True
indirim_onay = kupon_aktif and (yas >= 18 or toplam_sepet > 500)
print(indirim_onay)
```

Verilen kod satırları python kod bloku terminalde çalıştırıldığında ekrana basılacak olan mantıksal çıktı aşağıdakilerden hangisidir?

A) False
B) True
C) None
D) TypeError
E) ValueError

2. Bir listenin eleman sayısını döndüren yerleşik Python fonksiyonu hangisidir?

A) append()
B) insert()
C) len()
D) remove()
E) range()

3.

```
for sayi in range(1, 6):
    if sayi == 3:
        continue
    if sayi == 5:
        break
    print(sayi, end="-")
```

Yukarıda verilen kod bloku Python yorumlayıcısı tarafından çalıştırıldığında terminal ekranına basılacak standart veri çıktısı aşağıdakilerden hangisidir?

A) 1-2-
B) 1-2-4-
C) 1-2-3-4-
D) 1-2-4-5-
E) 1-2-3-4-5-

4. Bir yazılım projesinin belirli bir sürümünü etiketleyerek indirilebilir paket hâlinde yayımlama işlemine ne ad verilir?

A) Commit
B) Push
C) Shifting
D) Release
E) Installation

Ç) Aşağıdaki soruların cevaplarını teknik terimleri kullanarak açıklayınız.

1.

```
sayac = 1
while sayac <= 3:
    girilen_parola = input("Sistem parolasını giriniz: ")
    if girilen_parola == "Malazgirt1071":
        print("Giriş Başarılı.")
```

Yukarıda verilen kod bloku, bir sisteme giriş aşamasındaki hatalı parola denemelerini kontrol etmek amacıyla kurgulanmıştır. Buna göre a ve b'de yer alan soruları cevaplayınız.

a) Bu kod bloku, bu hâliyle çalıştırıldığında neden bir sonsuz döngü riski taşımaktadır? Python yorumlayıcısının çalışma mantığına göre açıklayınız.

b) Bu kodun kararlı çalışması ve kullanıcıya tam 3 hak tanıması için hangi satırın / satırların eklenmesi gerektiğini yazınız.

2. Kullanıcıdan input() ile bir parola alarak, for döngüsü yardımıyla bu parolanın her bir karakterini satır satır ekrana basan Python kodu nedir? Bu kodu, PEP 8 standartlarına uygun olarak yazınız.

ÖLÇME VE DEĞERLENDİRME

3.

```
# 1. Döngü
for i in range(5):
    print(i)
# 2. Döngü
for j in range(10, 2, -2):
    print(j)
```

Yukarıda iki farklı for döngüsü yapısı verilmiştir. Döngülerin Python yorumlayıcısında üreteceği sayılar ve terminale basılacak olan çıktılar nelerdir? Sırasıyla yazınız.

4. Bir bankamatik sisteminde kullanıcıya en fazla 3 kez PIN kodu girme hakkı tanınır. Kullanıcı ilk veya ikinci denemesinde PIN kodunu doğru girerse sistemin kalan hakları bekletmeden döngüyü anında kırması ve programı sonlandırması istenir. while döngüsünü kullanarak, kullanıcı pini 1453'e eşit olduğu doğrulaması sağladığı an "Erişim İzni Verildi." mesajı basıp döngüden çıkan Python kodu nedir? Yazınız.

5.

```
for sayi in range(1, 6):
    if sayi == 3:
        continue
    print(sayi)
```

Verilen kod çalıştırıldığında terminale basılacak çıktıyı yazınız ve continue ifadesinin döngü akışına olan etkisini bu kod üzerinden açıklayınız.

ÖLÇME VE DEĞERLENDİRME

6. Bir Kod-Asistan projesinde, sistemde kayıtlı olan misafir kullanıcıların isimleri bir liste içerisinde tutulmaktadır. Bu listenin elemanlarına for döngüsü ile erişip, her bir ismin başına "Merhaba" ekleyerek (örneğin Merhaba Musa) ekrana yazdıran Python kodu nedir? Yazınız.

```
kullanicilar = ["Musa", "Zeynep", "Harun", "Yusuf", "Ömer"]
```

7.

```
for i in range(3):
    print(f"Dış Döngü: {i}")
    for j in range(2):
        print(f"--- İç Döngü: {j}")
```

Yukarıda verilen koda göre a ve b sorularını cevaplayınız.

a) Verilen kod bloku toplamda kaç tur çalışır ve kaç satır çıktı üretir?

b) Kodun üreteceği ekran çıktısı nedir? Sırasıyla yazınız.

8.

```
vki_degerleri = [22.4, 16.2, 28.1]
for vki in vki_degerleri:
    if vki < 18.5:
        print("Filtre: Standart Değer Altı")
    elif 18.5 <= vki < 25:
        print("Filtre: Standart Değer")
    else:
        print("Filtre: Standart Değer Üstü")
```

Yukarıda verilen koda göre yorumlayıcının vki_degerleri listesindeki verileri yukarıdan aşağıya işleme kronolojisini dikkate alarak kodun üreteceği terminal çıktısını sırasıyla yazınız.

9. 1'den 15'e kadar olan sayıları ekrana basan, 3'e tam bölünen sayıları "3'e tam bölünür." mesajıyla veren Python kodu nedir? Yazınız.



Öğrenme Birimi Karekodu

4. ÖĞRENME BİRİMİ
MODÜLER PROGRAMLAMA VE FONKSİYONLAR

KONULAR

4.1. FONKSİYON TANIMLAMA VE VERİ AKIŞI
4.2. MODÜLER MİMARİ VE DEĞİŞKEN ETKİ ALANI
4.3. KOD KALİTESİ VE DENETİMİ

ANAHTAR KAVRAMLAR

- argüman
- değer döndürme (return)
- DRY prensibi
- etki alanı (scope)
- fonksiyon
- küresel değişken
- modülerlik
- parametre
- refactoring
- yerel değişken
- yerleşik fonksiyon

BEKLENTİLER

Bu öğrenme biriminde sizden

- fonksiyon yapılarını ve veri akışını standartlara göre tasarlamanız
- modüler mimariyi ve değişken etki alanını yönetmeniz
- yapay zekâ ve dijital araçlarla kod kalitesini denetlemeniz

beklenmektedir.

BAŞLARKEN

Büyük bir fabrikada tek bir işçinin bütün ürünleri tek başına oluşturmaya (montaj, boyama, paketleme vb.) çalıştığını düşününüz. Ne kadar yorucu ve hataya açık bir üretim süreci olurdu, değil mi? Bunun yerine her işçinin veya robotun kendi uzmanlık alanındaki tek bir göreve odaklanarak çalıştığı düzenli bir sistem düşününüz. Herkes kendi görevini kusursuz yaptığında ortaya kaliteli ve hatasız bir ürün çıkar. Peki, bir programdaki uzun kod satırları bu fabrikada olduğu gibi daha küçük, belirli görevleri yerine getiren ve yeniden kullanılabilir parçalara ayrılırsa ne değişir? Bir programda kodları görevlerine göre ayırmak hem programı okumayı kolaylaştırır hem de aynı işlemleri yeniden kullanmayı sağlar.

Bu öğrenme biriminde karmaşık kod yığınlarını fonksiyon adı verilen yetenekli görevlilere dönüştüreceksiniz. Kod-Asistan projenizi daha düzenli, yeniden kullanılabilir ve modüler bir yapıya dönüştürmeye hazır mısınız?

HAZIRLIK ÇALIŞMALARI

1. Kapsamlı bir okul gezisi düzenlenirken tüm organizasyon sorumluluğunun sadece tek bir kişiye verilmesi ne tür krizlere yol açabilir? Bu durumdan yola çıkarak; karmaşık bir bilgisayar oyunundaki kodları tek bir dosya yerine hareket, ses, grafik gibi alt parçalara bölerek yönetmenin yazılımcılara ne tür kolaylıklar sağlayacağını tartışınız.
2. Verilen anahtar kavramın size neleri çağrıştırdığını düşününüz. Bu kavramla ilgili aklınıza gelen en fazla 5 kelimeyi ilgili boşluklara yazınız. Ardından bu kelimeleri kullanarak anahtar kavramı açıklayan kısa bir cümle kurunuz.

Anahtar Kavram: İŞ BÖLÜMÜ

a)

b)

c)

ç)

d)

İlgili Cümlem:

.....

.....

4.1. FONKSİYON TANIMLAMA VE VERİ AKIŞI

Fonksiyonlar; programlama dillerinde belirli bir görevi yerine getirmek üzere yapılandırılmış, tekrar kullanılabilir kod bloklarıdır. Bir fonksiyonun tanımlanması, karmaşık kod yapılarını daha modüler hâle getirirken girdi ve çıktı mekanizmaları sayesinde verinin yazılım içinde düzenli bir şekilde akmasını sağlar. Yazılım mimarisinde işlevselliği artıran bu yapılar, geliştiriciye hem esneklik sunar hem de kodun okunabilirliğini optimize eder.

Modüler programlamada sıfırdan yazılan **kullanıcı tanımlı fonksiyonların (user-defined functions)** yanı sıra Python dil mimarisinin çekirdeğinde hazır olarak bulunan **yerleşik fonksiyonlar (built-in functions)** da mevcuttur. Bu fonksiyonlar, arka planda python yorumlayıcısı ve işletim sistemi aracılığıyla çok karmaşık veri akışlarını yürütmelerine rağmen yazılımcıya sadece basit bir isim ve parantez () yapısıyla sunulurlar. print(), input() gibi fonksiyonlar Python dilinin yerleşik fonksiyonlarından bazılarıdır.

1. ÖRNEK

YERLEŞİK (BUILT-IN) FONKSİYONLARLA VERİ AKIŞI

Kod-Asistan projesinin güvenlik denetimi aşamasında, kullanıcının sisteme girdiği bir onay kodunun kaç karakterden oluştuğunun ve bellekte hangi türde saklandığının analiz edilip ekrana yansıtılması gerekmektedir. Bir yazılımcının bu işlemi yapabilmesi için klavye donanımıyla veya RAM adresleriyle sıfırdan haberleşecek kodları yazması devasa bir zaman ve kodlama maliyeti yaratır. Bu donanımsal yük problemini aşmak ve veriyi saniyeler içinde işlemek için Python dil mimarisinde hazır olarak bekleyen input(), len(), type() ve print() yerleşik fonksiyonları birbiriyle haberleştirilerek bu sorun çözülecektir.

Örnek Kod Bloku

```
print("--- KOD-ASİSTAN SİSTEM DENETİMİ ---")
# input() yerleşik fonksiyonu, akışı durdurur ve dinamik veri alır.
sistem_kodu = input("Lütfen sistem onay kodunu giriniz: ")
# len() yerleşik fonksiyonu, verinin boyutunu (karakter sayısını) ölçer.
kod_uzunlugu = len(sistem_kodu)
# type() bellek türünü analiz ederken, print() sonucu ekrana yansıtır.
print(f"Girdiğiniz kod {kod_uzunlugu} karakterden oluşmaktadır.")
print("Bellekteki veri türü:", type(sistem_kodu))
```

Terminal Çıktısı

```
--- KOD-ASİSTAN SİSTEM DENETİMİ ---
Lütfen sistem onay kodunu giriniz: KOD1453
Girdiğiniz kod 7 karakterden oluşmaktadır.
Bellekteki veri türü: <class 'str'>
```

Kod Analizi

- input() fonksiyonu, standart girişten kullanıcının yazdığı satırı alır ve str türünde döndürür.
- len() fonksiyonu, ilgili nesnenin uzunluk bilgisini döndürür. Liste ve metin gibi temel yapılarda bu bilgi verimli biçimde elde edilir.
- type() fonksiyonu, kendisine verilen değerin hangi sınıfa ait olduğunu gösterir.
- print() fonksiyonu sonucu standart çıktı ortamında görüntüler.

Python'un yerleşik (built-in) fonksiyonları, arka planda çalışan karmaşık kod bloklarını basit bir isim ve parantez () arkasına gizleyerek programlama sürecini büyük ölçüde kolaylaştırır. Bazen Python'un içinde, karşılaşılan özel bir ihtiyacı giderecek hazır bir fonksiyon bulunmayabilir. Örneğin kullanıcının sepet tutarına KDV ekleyip net faturayı bulan ya da iki farklı dosyayı birleştirip şifreleyen bir yerleşik komut bulunmaz. Bu durumda Python geliştiricilerinin yaptığı gibi sisteme özel bağımsız fabrikaların (kullanıcı tanımlı fonksiyonlar) inşa edilmesini öğrenme aşamasına geçilir.

4.1.1. Kullanıcı Tanımlı Fonksiyonlar

Yazılım projeleri büyüdükçe aynı kod bloklarını (örneğin kullanıcıyı selamlama, yaş hesaplama veya menü gösterme) programın farklı yerlerinde defalarca kullanma ihtiyacı doğar. Bu kodları her defasında kopyalayıp yapıştırmak; kodun boyutunu (şişmesini) artırır, okunabilirliğini düşürür ve bir mantık hatası yapıldığında tüm kopyaların tek tek düzeltilmesini gerektirir. Python programlama dilinde tekrar eden kodları modüler ve sürdürülebilir yapılara dönüştürmek için fonksiyonlar (functions) kullanılır.

Fonksiyonlar, belirli bir görevi yerine getirmek üzere adlandırılmış ve gerektiğinde çağrılabilen kod bloklarıdır. Tıpkı modern bir kahve makinesi gibi çalışır. Düğmesine basılmadığı sürece çalışmaz (çağırma / tetikleme), içine kahve ve su konulduğunda bunları işler (parametre), bir fincan kahve sunar (return). Bazı fonksiyonlar hesaplanan bir değeri return ile döndürürken bazıları yalnızca belirli bir işlemi gerçekleştirir.

4.1.2. Fonksiyon Tanımlama Standartları (def Anahtar Kelimesi)

Python dünyasında özel fonksiyonlar oluşturulurken **def [define (tanımla)]** anahtar kelimesi kullanılır. Bu tanımlama yapılırken yazılan programın sorunsuz çalışması ve diğer yazılımcılar tarafından kolayca okunabilmesi için bazı altın kurallara dikkat edilmelidir.

Değişkenlerde olduğu gibi fonksiyon isimleri de Python topluluğunda yaygın kabul gören PEP 8 stil kılavuzuna uygun olarak snake_case notasyonuyla yazılmalıdır. Yani isimler tamamen küçük harflerden oluşmalı ve kelimeler birbirine alt çizgi (_) ile bağlanmalıdır. Ayrıca fonksiyon adlarının yaptığı işi açıkça anlatan, kısa ve anlamlı ifadelerden seçilmesi önerilir.

İsimlendirme yapıldıktan hemen sonra, fonksiyonun çalışabilmesi için herhangi bir başlangıç bilgisine (veriye) ihtiyaç duyup duymadığını gösteren parantezler () eklenir. Ardından, fonksiyonun yapacağı görevlerin başladığını bilgisayara bildiren iki nokta (:) sembolü konulur.

Fonksiyonun içine yazılacak tüm görev satırlarının aynı hizada ve içeriden yazılması gerekir. Python'da bu içeri kaydırma işlemi için standart olarak 4 boşluk kullanılır. Burada dikkat edilmesi gereken en önemli PEP 8 stil önerisi, boşluk tuşu ile Tab tuşunun kesinlikle birbirine karıştırılmamasıdır. Aksi takdirde Python yorumlayıcısı girinti hatası (IndentationError) verir ve kod çalışmaz. PEP 8 stil önerilerine ve girinti kurallarına uygun temel bir fonksiyon tanımlama şablonu aşağıda gösterilmiştir:

```
# Fonksiyon Tanımlama Şablonu
def fonksiyon_adi(parametre_1, parametre_2):
    # Fonksiyon Gövdesi (4 boşluk girinti disiplini)
    islem_sonucu = parametre_1 + parametre_2
    return islem_sonucu
```

2. ÖRNEK

PARAMETRESİZ TEMEL FONKSİYON VE ÇAĞIRMA (CALL)

Kod-Asistan projesinin başlangıcında kullanıcıyı karşılayan standart selamlama mesajının, kodun içinde defalarca yazılmak yerine merkezî bir fonksiyona dönüştürülmesi ve ihtiyaç anında çağrılması test edilecektir.

Örnek Kod Bloku

```
# 1. AŞAMA: Fonksiyonun Tanımlanması (Makinenin Kurulumu)
def asistan_karsilama():
    print("=== Kod-Asistan v1.0 Sistemine Hoş Geldiniz ===")
    print("Siber Güvenlik Protokolleri Aktif.")
    print("-----------------------------------------------")
# 2. AŞAMA: Fonksiyonun Çağrılması (Makinenin Çalıştırılması)
# Fonksiyon tanımlandığında kendi kendine çalışmaz. İsmiyle çağrılması (call) gerekir.
asistan_karsilama()
asistan_karsilama()
```

Terminal Çıktısı

```
=== Kod-Asistan v1.0 Sistemine Hoş Geldiniz ===
Siber Güvenlik Protokolleri Aktif.
-----------------------------------------------
=== Kod-Asistan v1.0 Sistemine Hoş Geldiniz ===
Siber Güvenlik Protokolleri Aktif.
```

Kod Analizi

- def asistan_karsilama(): satırı ile fonksiyon tanımlanmış ancak henüz çağrılmadığı için gövdesindeki kodlar çalıştırılmamıştır.
- Girinti bitip ana akışa dönüldüğünde, asistan_karsilama() yazılarak fonksiyon iki kez çağrılmış ve her çağrıda gövdesindeki kodlar çalıştırılmıştır.

4.1.3. Parametre Yönetimi ve Veri Aktarımı (Girdi)

Fonksiyonlar, kendilerine verilen görevi yerine getirebilmek için genellikle bazı başlangıç bilgilerine ihtiyaç duyar. Örneğin bir toplama fonksiyonunun işlem yapabilmesi için öncelikle hangi sayıları toplayacağını bilmesi gerekir. Bir fonksiyona işlem yapması için gönderilen bu verilere **argüman**, ilk kez oluşturulan (tanımlanan) fonksiyonun hangi verilerle çalışacağının belirtildiği şablon değişkenlere ise **parametre** adı verilir. Parametreler, fonksiyonun sadece kendi içinde geçerli olan yerel değişkenleridir ve aynı fonksiyonun farklı değerlerle tekrar tekrar kullanılabilmesini sağlar.

Programlama dünyasında sıkça karıştırılan bu iki kavramın birbirinden ayrılması oldukça kolaydır. Parametre, fonksiyon tanımlanırken parantez içine yazılan ve fonksiyona gönderilecek değeri temsil eden yerel addır (örneğin parametre_1 ve parametre_2). Argüman ise fonksiyon program içinde çalıştırılırken (çağrılırken) parantez içine gönderilen ve parametre şablonunun yerini dolduran somut ve gerçek veriler olarak tanımlanır. Örneğin fonksiyon_adi(25, 40) şeklinde bir komut yazıldığında parantez içindeki 25 ve 40 değerleri fonksiyona gönderilen birer argümandır.

3. ÖRNEK

PARAMETRE ALAN FONKSİYON

Sistemdeki kullanıcıyı sadece genel bir mesajla değil doğrudan kendi adıyla ve yetki rolüyle karşılayan parametreli bir selamlama katmanı tasarlamanız istenmektedir.

Örnek Kod Bloku

```
# ‘kullanici_adi’ ve ‘rol’ dışarıdan gelecek verileri tutan parametrelerdir.
def kisisellestirilmis_karsilama(kullanici_adi, rol):
    print(f"Merhaba {kullanici_adi}. Sisteme ‘{rol}’ yetkisiyle bağlandınız.")

# Fonksiyon çağrılırken parantez içine gönderilen gerçek verilere "Argüman" denir.
kisisellestirilmis_karsilama("Fatih Sultan Mehmet", "Yönetici")
kisisellestirilmis_karsilama("Yahya", "Misafir")
```

4.1.4. Dönüş Değeri İfadesinin Veri Aktarma Rolü (Çıktı)

Bir fonksiyonun ekrana print() ile yazı yazdırması, o verinin programın geri kalanı tarafından kullanılabileceği anlamına gelmez. print() değerin metinsel gösterimini ekrana yazar ve anlamlı bir sonuç yerine None döndürür. Hesaplanan değerin başka işlemlerde kullanılabilmesi için return kullanılmalıdır.

Bir fonksiyonun yaptığı hesaplamanın veya işlemin sonucunu, programın ana akışına (çağrıldığı yere) geri göndermesi için **return (döndür)** anahtar kelimesi kullanılır. return ifadesi, fonksiyonun işini bitirdiğini ve veriyi teslim ettiğini (kahve makinesinin kahveyi bardağa dökmesi gibi) simgeler.

4. ÖRNEK

BİRİM DÖNÜŞTÜRÜCÜ MODÜLÜ (RETURN İLE VERİ AKTARIMI)

Problem Senaryosu: Kod-Asistan projesinin dışarıdan alınan inç (inch) cinsinden bir ölçüyü santimetreye çeviren bir dönüştürücü modülüne ihtiyaç duyulmaktadır. Bu modülün görevi sadece ekrana yazı yazdırmak değil hesapladığı santimetre değerini çağrıldığı yere döndürerek (return) sonraki mantıksal karar aşamalarında (örneğin 50 cm sınır kontrolü) kullanılabilmesini sağlamaktır.

Örnek Kod Bloku

```
def inc_to_cm(inc_degeri):
    """
    Aldığı inç değerini 2.54 sabiti ile çarparak santimetreye dönüştürür.
    Elde edilen sonucu ekrana basmaz, çağrıldığı yere döndürür (return).
    """
    # 1 inç = 2.54 cm'dir.
    cm_sonuc = inc_degeri * 2.54
    return cm_sonuc
# --- FONKSİYONUN ANA AKIŞ İÇERİSİNDE ÇAĞRILMASI (MAIN FLOW) ---
print("--- KOD-ASİSTAN DONANIM ÖLÇÜM DÖNÜŞTÜRÜCÜ ---")
# Kullanıcıdan dönüştürülecek inç verisinin dinamik olarak alınması.
tasarim_inci = float(input("Lütfen dönüştürülecek ölçüyü giriniz (İnç): "))
# Fonksiyon çağrılıyor (call) ve return edilen veri 'hesaplanan_cm' değişkenine aktarılıyor.
hesaplanan_cm = inc_to_cm(tasarim_inci)
# Ana akışa aktarılan veri, artık sistemin istenen yerinde özgürce kullanılabilir.
print(f"Dönüşüm Tamamlandı: {tasarim_inci} inç = {hesaplanan_cm:.2f} cm'dir.")
# Veri aktarımının kanıtı: Döndürülen santimetre değeri başka bir koşulun denetlenmesinde
kullanılır.
if hesaplanan_cm > 50:
    print("Sistem Uyarısı: Bu donanım boyutu standart asistan paneli için büyüktür.")
```

Terminal Çıktısı

```
--- KOD-ASİSTAN DONANIM ÖLÇÜM DÖNÜŞTÜRÜCÜ ---
Lütfen dönüştürülecek ölçüyü giriniz (İnç): 22.5
Dönüşüm Tamamlandı: 22.5 inç = 57.15 cm'dir.
Sistem Uyarısı: Bu donanım boyutu standart asistan paneli için büyüktür.
```

4. ÖRNEK

Kod Analizi

- Fonksiyon çağrıldığında (inc_to_cm(tasarim_inci)), ana akıştan gönderilen veri fonksiyonun yerel parametresine (inc_degeri) aktarılır.
- Fonksiyon kendi içinde yerel bir değişken (cm_sonuc) yaratarak aritmetik dönüşümü tamamlar.
- Hesaplanan sonuç return komutuyla ana akışa döndürülür ve fonksiyonun o turdaki görevi sonlanır.

Veri Akışı Elle Çalıştırma (Dry Run) Tablosu: Tablo 4.1'de, terminal çıktısında verilen 22.5 inçlik örnek girdi üzerinden fonksiyonun bellek adımları satır satır modellenmiştir.

Tablo 4.1: Fonksiyonun Bellek Adımlarının Modellenmesi

Adım	Akış Yönü	Olay / İşlem	inc_degeri (Parametre)	cm_sonuc (Yerel Değişken)	return Edilen (Çıkan) Değer
1	Ana akıştan fonksiyona	inc_to_cm(22.5) çağrıldı. 22.5 argümanı fonksiyonun parametresine aktarıldı.	22.5	-	-
2	Fonksiyon içi	22.5 * 2.54 işlemi Python yorumlayıcısı tarafından çalıştırıldı.	22.5	57.15	-
3	Fonksiyondan ana akışa	return cm_sonuc çalıştı. 57.15 değeri fonksiyondan ana akışa fırlatıldı.	22.5	57.15	57.15 (float)
4	Ana akış	return edilen 57.15 verisi ana programdaki hesaplanan_cm değişkenine atanmıştır.	-	-	-

return anahtar kelimesi sadece bir veriyi dışarı döndürmekle (yazılım jargonundaki yaygın kullanımıyla veriyi ana akışa fırlatmakla) kalmaz, aynı zamanda return çalıştığında fonksiyonun yürütülmesi sona erer ve varsa döndürülen değer çağrıldığı yere iletilir. Bu nedenle fonksiyon gövdesi içinde yalnızca çalıştırılmış bir return ifadesinden sonra aynı yürütme yolunda kalan satırlara erişilemez. Fonksiyon içinde yazılan ama return yüzünden hiçbir zaman çalıştırılamayacak olan bu işlevsiz satırlara yazılım literatüründe **ölü kod** adı verilir.

1. UYGULAMA

Uygulamanın Adı: KOD-ASİSTAN SELAMLAMA VE HESAPLAMA MODÜLÜ (PARAMETRE VE RETURN KULLANIMI)

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: Tekrar eden kodları fonksiyonlar kullanarak modüler ve sürdürülebilir yapılara dönüştürmek, parametre yönetimi ve return ifadesinin veri aktarma rolünü sarmal proje üzerinde yapılandırmak

Problem Senaryosu: Kod-Asistan projenizin önceki (v0.3) sürümünde kullanıcıyı selamlama ve kullanıcının yaşını hesaplama işlemleri ana akış içerisine dağınık hâlde yazılmıştı. Bu durum, aynı işlem başka bir menüde tekrar istendiğinde kod tekrarına yol açmaktadır. Projenin yeni sürümüne giden yolculuğunda asistanınıza modüler bir kimlik kazandırmak için bu işlemleri parametre alan ve geriye sonuç döndüren bağımsız fonksiyonlara ayırmanız istenmektedir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel şekilde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Kod Dosyasının Hazırlanması ve Fonksiyonların Tanımlanması

a) Kod editörünü açarak kod_asistan_fonksiyonlar.py isimli yeni bir dosya oluşturunuz.

b) Kullanıcının adını ve rolünü (Yönetici / Misafir) parametre olarak dışarıdan alan, bu verileri kullanarak f-string ile kişiselleştirilmiş bir selamlama metni üreten ancak **ekrana yazdırmak yerine** selamlama metnini fonksiyonun çağrıldığı yere **return ile döndüren** karsilama_modulu(ad, rol) fonksiyonunu def anahtar kelimesiyle tanımlayınız.

c) Kullanıcının doğum yılını ve mevcut yılı parametre olarak alan, yaklaşık yaş hesaplamasını yaparak sonucu return eden yas_hesapla(dogum_yili, mevcut_yil) fonksiyonunu 4 boşluklu girinti kurallarına uyarak kurgulayınız.

2. Aşama: Ana Akışın Kurulması ve Fonksiyonların Çağrılması

a) Fonksiyonların dışına çıkarak ana akışı kurgulayınız.

b) Kullanıcıdan input() fonksiyonu ile ad, rol ve doğum yılı verilerini dinamik olarak alınız ve doğru veri türlerine dönüştürünüz.

c) Yazdığınız iki bağımsız fonksiyonu, kullanıcıdan aldığınız bu gerçek verileri (argümanları) parantez içine yazarak çağırınız (call) ve return edilen sonuçları yeni değişkenlere kaydedip print() ile ekrana yazdırınız.

1. UYGULAMA

```
# ========================================================================
# PROJE ADI: KOD-ASİSTAN (Sürüm: v0.4 - Modüler Dönüşüm)
# ETKİNLİK: SELAMLAMA VE HESAPLAMA MODÜLÜ (PARAMETRE VE RETURN)
# ========================================================================
# 1. AŞAMA: MODÜLER FONKSİYONLARIN TANIMLANMASI
def karsilama_modulu(ad, rol):
    """
    Kullanıcının ad ve rol bilgilerini parametre olarak alıp,
    kişiselleştirilmiş selamlama metnini return ile fırlatır.
    """
    # f-string ile bellekte metin biçimlendirme (Ekrana basılmaz, döndürülür.)
    selamlama_metni = (f"Hoş geldiniz {ad}. Sistemdeki rolünüz: [{rol}]. Kod-Asistan sizi
selamlıyor!")
    return selamlama_metni
def yas_hesapla(dogum_yili, mevcut_yil):
    """
    Doğum yılı ve mevcut yıl parametrelerini alarak yaklaşık yaş hesabı yapar ve elde
edilen tam sayı sonucu return eder.
    """
    # 4 boşluklu girinti kuralına uygun hesaplama bloku
    hesaplanan_yas = mevcut_yil - dogum_yili
    return hesaplanan_yas
# 2. AŞAMA: ANA AKIŞIN (MAIN FLOW) KURULMASI VE FONKSİYONLARIN ÇAĞRILMASI
# Kullanıcıdan dinamik veri alımı ve veri türü tuzaklarının engellenmesi
print("--- KOD-ASİSTAN KULLANICI GİRİŞ PANELİ ---")
giris_adi = input("Lütfen adınızı giriniz: ")
giris_rolu = input("Sistem rolünüzü giriniz (Yönetici/Misafir): ")
# Tür dönüşümü (Type casting) yapılarak input verisi int formatına dökülüyor.
dogum_tarihi = int(input("Lütfen doğum yılınızı giriniz (Örn: 2005): "))
guncel_yil = 2026  # Örnek sabit yıl. İstenirse datetime modülüyle dinamik alınabilir.
print("\n--- SİSTEM RAPORU VE ÇIKTILAR ---")
# karsilama_modulu fonksiyonunun çağrılması (call) ve sonucun yakalanması
asistan_mesaji = karsilama_modulu(ad=giris_adi, rol=giris_rolu)
print(asistan_mesaji)  # Return edilen veri burada ekrana yazdırılıyor.
# yas_hesapla fonksiyonunun çağrılması ve sonucun yakalanması
kullanici_yasi = yas_hesapla(dogum_yili=dogum_tarihi, mevcut_yil=guncel_yil)
print(f"Sistem tarafından hesaplanan güncel yaşınız: {kullanici_yasi}")
```

1. UYGULAMA

3. Aşama: Projenin Dijital Depoya Yüklenmesi

a) Kararlı çalışan fonksiyonel kodunuzu kaydedip dijital deponuza yükleyiniz.

b) Sürüm notu (Commit message) alanına "Kod-Asistan selamlama ve hesaplama adımları parametre ve return içeren modüler fonksiyonlara dönüştürüldü." yazarak deponuzu güncelleyiniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

4.1.5. Argüman Aktarım Yöntemleri ve Pozisyonel Eşleşme

Fonksiyonlar çağrılırken parametrelere gönderilen argümanların hangi kurala göre eşleşeceği, veri akışının kararlılığı açısından kritik bir öneme sahiptir. Python dil mimarisinde en temel argüman aktarım yöntemi **pozisyonel argümanlar (positional arguments)** kuralıdır.

Pozisyonel eşleşmede yorumlayıcı; fonksiyon çağrılırken yazılan argümanları, fonksiyon tanımlanırken belirlenen parametrelerle **yazılış sırasına göre (soldan sağa)** birebir eşleştirir.

Sıralamanın belirleyici olduğu bu yapı, sınavlarda kullanılan optik formların değerlendirilme mantığıyla büyük bir benzerlik gösterir. Bir optik okuyucu makine, kâğıt üzerindeki işaretlemeleri anlamlarına göre değil sadece kodlandıkları konuma (pozisyona) göre okur. Eğer bir öğrenci dalgınlıkla yan yana bulunan T.C. Kimlik Numarası bölümüne okul numarasını, Okul Numarası bölümüne ise T.C. kimlik numarasını kodlarsa makine bu durumu sorgulamaz veya insan zekâsıyla mantık yürütüp düzeltmez. Her iki alan da sayısal olduğu için makine bu verileri tam olarak kodlandıkları sıraya göre sisteme kaydeder. Bu donanımsal işlemde olduğu gibi yazılım dünyasında fonksiyon çağrılırken parantez içine yazılan değerler, hiçbir özellik belirtilmediği sürece sadece bulundukları sıraya (pozisyona) bakılarak bellekteki parametrelerle eşleştirilir. Bu katı sıralama kuralının kavranması, yazılımların mantıksal hatalar üretmeden kararlı bir şekilde çalışabilmesi için zorunludur. Eğer T.C. kimlik numarası ve okul numarası bekleyen bir fonksiyona argümanlar ters sırayla gönderilirse argümanlar uyumlu türlerdeyse program hata vermeden çalışabilir ancak değerler yanlış parametrelerle eşleştiği için mantıksal hata oluşabilir. Bu tür sessiz ve gizli hataların önüne geçmek için parametrelerin tanımlanma hiyerarşisine dikkat edilmesi önemlidir. Pozisyonel (sıralı) aktarımda argümanların bu sıraya kusursuz bir şekilde uyması zorunludur. Ancak anahtar sözcüklü (keyword) argümanlar kullanılarak bu sıra bağımlılığı azaltılabilir (Örneğin: karsilama_modulu(rol="Misafir", ad="Yahya")).

5. ÖRNEK

POZİSYONEL EŞLEŞME VE TÜR SAPMASI ANALİZİ

Kod-Asistan sisteminde, kullanıcının adını ve personel kimlik numarasını yazdıran fonksiyon tasarlanacaktır.

Örnek Kod Bloku

```
# İki parametreli bir fonksiyon tanımı
def kullanici_tescil(isim, personel_id):
    print(f"Kullanıcı: {isim} | ID: {personel_id}")
# DOĞRU ÇAĞRI: Argümanlar parametre sırasına (str, int) tam uyumlu gönderilmiştir.
kullanici_tescil("Fatih", 1453)
# HATALI ÇAĞRI (Mantıksal Sapma): Sıra karıştırılmıştır.
kullanici_tescil(1453, "Fatih")
# Çıktı: Kullanıcı: 1453 | ID: Fatih -> Program çökmez ancak veri akışı bozulur.
```

Kod Analizi

- İlk fonksiyon çağrısında argümanlar parametre sırasına (str, int) tam uyumlu gönderilmiştir.
- İkinci fonksiyon çağrısında sıra karıştırılmıştır.
- Program çökmemiş ama mantıksal hata meydana gelmiştir.

4.1.6. Veri Akışında Esneklik: Varsayılan Değerli Parametreler

Bazı senaryolarda, bir parametreye dışarıdan her seferinde argüman göndermek zorunlu olmayabilir. Fonksiyon çağrılırken belirli bir parametre için değer gönderilmediğinde Python yorumlayıcısının hata vermeden varsayılan bir sabit değeri kabul etmesi istenebilir. Bu duruma **varsayılan değerli parametreler (default parameters)** adı verilir.

Varsayılan değer ataması, fonksiyon tanımlanırken (def satırında) ilgili parametreye atama operatörü (=) kullanılarak doğrudan verilir. Python dil mimarisinde, varsayılan değerli parametreler tanımlanırken her zaman parantezin **en sonuna (sağına)** yazılmak zorundadır. Varsayılan değerli bir parametreden sonra varsayılan değeri olmayan normal bir parametre yazılamaz. Bu kuralın ihlal edilmesi durumunda Python SyntaxError üretir ve program çalıştırılamaz.

6. ÖRNEK

KOD-ASİSTAN MODÜLER BULUT DEPOLAMA MALİYET HESAPLAYICISI

Kod-Asistan projenizin v0.5 sürümünde, kullanıcıların bulut üzerinde yedekleyeceği veriler için dinamik bir maliyet hesaplama modülü entegre etmeniz istenmektedir. Sistem mimarisinde, sunucu ve bant genişliği maliyetleri için standart bir indirim oranı belirlenmiştir ancak bazı kurumsal veya kıdemli kullanıcılara (Yöneticiler vb.) daha yüksek indirimler tanımlanabilmelidir. Eğer fonksiyon çağrılırken özel bir indirim oranı belirtilmezse sistem otomatik olarak varsayılan indirim oranını devreye almalıdır.

BİLGİ: Yazılım projelerinde para hesaplamaları yaparken float veri türü yaklaşık sonuçlar üretebilir. Bu örnek eğitim amaçlı basitleştirilmiştir, gerçek finansal uygulamalarda "Decimal" modülü gibi daha hassas yöntemler tercih edilir. Ayrıca gerçek bir faturada, uygulamanın kararlı kalabilmesi için tutar ve indirim_orani verilerinin negatif girilmesini veya indirimin %100'ü aşmasını engelleyecek geçerlilik kontrolleri bulunmalıdır.

Örnek Kod Bloku

```
# ==========================================================================
# PROJE ADI: KOD-ASİSTAN (Sürüm: v0.5 - Akıllı Maliyet Yönetimi)
# ÖRNEK: BULUT DEPOLAMA FATURA MODÜLÜ (VARSAYILAN PARAMETRE)
# ==========================================================================
def fatura_kes(tutar, indirim_orani=10):
    """
    Bulut depolama ücretini hesaplar.
    indirim_orani parametresi boş bırakılırsa varsayılan olarak 10 kabul edilir.
    """
    # İndirimli net tutarın matematiksel olarak hesaplanması
    net_tutar = tutar - (tutar * indirim_orani / 100)
    return net_tutar
# --- FONKSİYONUN ANA AKIŞ İÇERİSİNDE TEST EDİLMESİ ---
print("--- KOD-ASİSTAN BULUT MALİYET RAPORU ---")
# 1. SENARYO: Kurumsal / Kıdemli Kullanıcı (Özel İndirim Oranı Gönderildi)
# İkinci argüman olarak 20 gönderildiği için varsayılan değer (10) ezilir.
kurumsal_fatura = fatura_kes(1000, 20)
print(f"1. Senaryo (Yönetici İndirimi %20) -> Ödenecek Tutar: {kurumsal_fatura:.2f} TL")
# 2. SENARYO: Standart / Misafir Kullanıcı (Tek Argüman Gönderildi)
# İkinci argüman boş bırakıldığı için fonksiyon otomatik olarak indirim_orani = 10 alır.
standart_fatura = fatura_kes(1000)
print(f"2. Senaryo (Standart İndirim %10) -> Ödenecek Tutar: {standart_fatura:.2f} TL")
```

6. ÖRNEK

Terminal Çıktısı

--- KOD-ASİSTAN BULUT MALİYET RAPORU ---
1. Senaryo (Yönetici İndirimi %20) -> Ödenecek Tutar: 800.00 TL
2. Senaryo (Standart İndirim %10) -> Ödenecek Tutar: 900.00 TL

Kod Analizi

- 1. senaryoda fatura_kes(1000, 20) komutu çalıştırıldığında Python yorumlayıcısı sırasıyla parametreleri eşleştirir. tutar hücresine 1000, indirim_orani hücresine ise gönderilen 20 değerini yazar. Bir diğer ifadeyle ikinci argüman gönderildiğinde o çağrı için indirim_orani parametresine gönderilen değer atanır.
- 2. senaryoda fatura_kes(1000) komutu verildiğinde ise yorumlayıcı ikinci bir argüman bulamaz. Varsayılan değeri bulunan indirim_orani parametresi için argüman gönderilmediğinde varsayılan değer kullanılır. Böylece fonksiyon imzasında bekleyen 10 değeri boş kalan hücreye atanır ve sadece bu parametrenin eksikliğinden kaynaklanabilecek TypeError istisnası önlenmiş olur (Tutar argümanı hiç verilmezse zorunlu bir parametre eksik bırakıldığı için program yine hata fırlatacaktır.).

Fonksiyon tasarımlarında varsayılan (default) değer atamak, bilgisayarın eksik veriyle karşılaştığında hata mesajı üretmesini önleyen ve veri akışına esneklik kazandıran temel bir mimaridir. Bir fonksiyona dışarıdan özel bir argüman gönderildiğinde bu yeni değer fonksiyonun tanımında yer alan varsayılan sabitin yerine geçer. Bu işleme yazılım literatüründe **değerin ezilmesi (overriding)** adı verilir. Dışarıdan herhangi bir veri gönderilmezse bilgisayar işlemi durdurmak yerine doğrudan belleğinde tuttuğu o varsayılan değeri işleme alır ve akışa kararlı bir şekilde devam eder. Bu esnek veri akışı, günlük hayatta bir restoranda verilen döner menü siparişlerinin işleyişiyle büyük bir benzerlik gösterir.

- Bir müşteri sadece "döner menü" talep ettiğinde, sistem otomatik olarak "ayran" ve "standart boy patates" seçeneklerini (varsayılan değerleri) devreye sokarak siparişi oluşturur. Müşteri yan ürünlerin detaylarını belirtmediği için restoran sistemi çökmez veya siparişi iptal etmez, doğrudan standart kuralları çalıştırarak hizmeti tamamlar.
- Müşteri sipariş esnasında "içecek şalgam olsun" veya "patatesler büyük boy olsun" şeklinde özel bir talep (argüman) sunarsa bu yeni bilgi sistemdeki standart seçeneği ezer ve sipariş fişi üzerine kullanıcının yeni tercihi kaydedilir.

Bu otomasyonda olduğu gibi Python dilinde fonksiyon çağrılırken parantez içine yazılan değerler mevcut varsayılan sabiti değiştirir. Parantez içi boş bırakıldığında ise sistem kendi standart atamasıyla yoluna devam eder. Bu mimarinin kavranması, özellikle farklı ihtimallere ve kullanıcı tercihlerine açık algoritmaların kurgulanmasında kritik bir rol oynar. Sistemin hata toleransını artıran bu yapı sayesinde bilgisayar programları yüzlerce farklı ihtimal için ayrı ayrı kod satırları yazılmasına gerek kalmadan tek bir fonksiyon gövdesi üzerinden yönetilebilir. Kurumsal bir indirim oranı hesaplanırken veya bir oyun senaryosunda karakterin aldığı hasar değerlendirilirken varsayılan parametreler, yalnızca bazı argümanların isteğe bağlı olmasını sağlayarak koda kullanım esnekliği kazandırır.

2. UYGULAMA

Uygulamanın Adı: DİJİTAL OYUN MOTORU: KARAKTER HASAR MEKANİZMASI

Uygulama Süresi: 20 Dakika

Problem Tanımı: Bir dijital macera oyununun arka plan kodlarını (oyun motorunu) yazmaktasınız. Oyundaki kahramanlar, canavarlarla mücadele ettikçe belirli bir hasar almaktadır ancak bazı durumlarda kahramanlarımızın elinde koruyucu bir kalkan bulunabilir.

Oyun motorunda karakterin kalan canını hesaplayıp ekrana basacak mevcut_can, darbe_gucu ve kalkan adında üç parametreye sahip hasar_al adında esnek bir fonksiyon tasarlamanız istenmektedir. Eğer oyuncunun kalkanı yoksa (bir diğer ifadeyle fonksiyon çağrılırken kalkan belirtilmezse) kalkan varsayılan olarak 0 kabul edilmeli ve darbe gücü doğrudan candan düşülmelidir. Oyuncu kalkan puanı belirtirse kalkan değeri darbe gücünden düşülmeli (bir diğer ifadeyle darbe gücü kalkan kadar hafifletilmeli) ve kalan net hasar candan eksiltilmelidir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel şekilde gerçekleştiriniz.

Uygulama Adımları

- Kod editöründe macera_oyun_motoru.py adında temiz bir Python kaynak kod dosyası açınız.
- Üç parametre alan hasar_al fonksiyonunu tanımlayınız. Üçüncü parametreye varsayılan değer atamayı unutmayınız.
- Fonksiyonun içinde net hasarı hesaplayan mantığı kurunuz. Kalan canı hesaplayıp f-string teknolojisiyle ekrana yazdırınız.
- Fonksiyonunuzu ana programda şu iki popüler oyun senaryosuyla çağırarak test ediniz:

  1. Senaryo (Kalkansız-Yalın Hasar): hasar_al(100, 30) çağrısı yapıldığında ekranda "Kahraman darbe aldı! Kalkan yok. Kalan Can: 70" çıktısını görmelisiniz.

  2. Senaryo (Kalkan Aktif-Değeri Ezme): hasar_al(100, 30, 10) çağrısı yapıldığında kalkan darbeyi 10 puan hafifletmeli ve ekranda "Kahraman darbe aldı! Kalkan darbeyi emdi. Kalan Can: 80" çıktısı oluşmalıdır.

- **Yapay Zekâ ile Karşılaştırmalı Denetim**: Yazdığınız kodu bir yapay zekâya (LLM) gönderiniz ve ondan **Bu fonksiyonu, karakterin canı 0'ın altına düştüğünde ekrana 'Oyun Bitti' yazacak şekilde günceller misin?** talebinde bulununuz. Yapay zekânın yazdığı kodu inceleyin, if-else yapısını fonksiyon içine yerleştirme mantığını İnsan Denetimi süzgecinizden geçirerek analiz ediniz.
- Kodunuzun en üstüne yorum satırlarıyla bu algoritmayı tasarlarken kodlama aşamasında yardımlaştığınız takım arkadaşınızın ismini yazarak yazılım etiğine uygun davranınız.
- Kodunuzu terminalde test edip hatasız çalıştırdıktan sonra dijital deponuza "Oyun motoru hasar fonksiyonu dinamik parametrelerle entegre edildi." sürüm notuyla yükleyiniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

4.2. MODÜLER MİMARİ VE DEĞİŞKEN ETKİ ALANI

Bilgisayar programlarında tanımlanan değişkenler, Python yorumlayıcısı tarafından yürütülürken geçici bellek hücrelerinde depolanır. Yazılan bir değişkenin programın hangi satırlarından okunabileceğini ve hangi bloklarda geçerli olduğunu belirleyen kurallar hiyerarşisine **değişkenlerin etki alanı (variable scope)** adı verilir.

Python dil mimarisinde ve modüler programlama tasarımlarında değişkenler, bellekteki konumlarına ve erişim sınırlarına göre iki temel gruba ayrılır.

Küresel Değişkenler (Global Variables): Herhangi bir fonksiyon gövdesinin dışında, programın ana gövdesinde tanımlanan değişkenlerdir. En büyük avantajları, aynı modüldeki tüm fonksiyonların içinden ve dışından hiçbir kısıtlama olmaksızın doğrudan okunabilmeleridir.

Yerel Değişkenler (Local Variables): Bir fonksiyonun gövdesi içinde (4 boşluk girintili alanda) tanımlanan değişkenlerdir. Yerel değişkenler, sadece ait oldukları fonksiyon ana program tarafından çağrıldığında (Function Call) RAM üzerinde anlık olarak tahsis edilirler. Bu değişkenlerin ömrü ve görünürlüğü sadece fonksiyonun girintili iç alanıyla sınırlıdır. Fonksiyonun görevi return komutuyla bittiği an bu yerel değişkenler, bilgisayarın bellek temizleme mekanizması tarafından RAM'den tamamen silinir.

4.2.1. Bellekte İsim Çakışmaları ve Gizleme Kuralları

Bir programda bir küresel değişken ile bir yerel değişken aynı isme sahip olabilir. Bu durumda Python yorumlayıcısı, fonksiyonun içinde işlem yaparken her zaman **yerel değişkene öncelik verir**. Fonksiyon içindeki yerel değişken, aynı isimdeki küresel değişkenin değerini geçici olarak gölgeler. Buna yazılım literatüründe **gizleme (shadowing)** adı verilir.

Kurumsal bir iş yerinde aynı isme sahip çalışanlar arasında yaşanan iletişim süreçleri, bu gizleme mantığının çalışma hiyerarşisine verilebilecek somut bir örnektir. Bir şirketin genel müdürünün adının "Murat" (küresel değişken) ve donanım departmanında çalışan bir uzmanın adının da "Murat" (yerel değişken) olduğu varsayıldığında sistemde bir isim çakışması meydana gelir. Donanım departmanının sınırları (fonksiyon gövdesi) içerisinde "Bu dosyayı Murat'a iletin." talimatı verildiğinde çalışanlar doğrudan kendi departmanlarındaki uzmanı hedef alır ve işlemi ona göre tamamlar. Bu kapalı alanın içinde genel müdür olan Murat'ın kimliği geçici olarak arka planda kalır ve gölgelenir. Bu hiyerarşik iletişimde olduğu gibi Python dil mimarisinde bilgisayar belleği, isim çakışmalarını konum önceliğine göre çözer. Bir fonksiyonun içinde küresel değişkenle aynı isme sahip yeni bir yerel değişken tanımlandığında bilgisayar önceki küresel değeri silmez veya bozmaz. O fonksiyonun sınırları dâhilinde geçerli olmak üzere geçici bir yerel alan tahsis eder. Fonksiyonun görevi bittiğinde ise yerel değişken bellekten tamamen temizlenir. Ana programda aynı isim tekrar çağrıldığında, sistem doğrudan bellekte yaşamaya devam eden küresel değere (genel müdüre) ulaşarak veri akışını güvenli bir şekilde sürdürmeye devam eder.

3. UYGULAMA

Uygulamanın Adı: DEĞİŞKEN ETKİ ALANI VE BELLEK DENETİMİ

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: Programlama dilinde küresel ve yerel değişkenleri tanımlayarak bellek hiyerarşisindeki yerlerini saptamak, fonksiyon içinden küresel verilere erişim mantığını incelemek ve yerel değişkenlerin fonksiyon dışından çağrılmasıyla oluşan bellek sınır ihlallerini (NameError) analiz etmek

Problem Senaryosu: Kod-Asistan projenizin çekirdek sürüm ayarlarını (asistan adı, sistem sürümü vb.) her yerden erişilebilir küresel değişkenler olarak, sadece belirli bir raporlama anında gereken verileri ise yerel değişkenler olarak kurgulamanız istenmektedir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel şekilde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Kod Dosyasının Hazırlanması ve İncelenmesi

a) Kod editörünü açarak kod_asistan_bellek.py isimli yeni bir boş kod dosyası oluşturunuz.

b) Dosyanın en üstüne (küresel alana) asistan_adi ve sistem_surumu değişkenlerini yazarak bunlara sırasıyla **KODİ** ve **v1.2** metinsel değerlerini atayınız.

c) Alt satırlara geçerek def anahtar kelimesiyle asistan_raporu_bas() isimli bir fonksiyon başlatınız.

ç) Fonksiyonun gövdesine girerek (4 boşluk içeriden), sadece bu fonksiyon çalıştığında yaşayacak olan bellek_durumu = "Kararlı / RAM Optimize Edildi" yerel değişkenini kodlayınız.

d) Fonksiyonun devamına, f-string teknolojisini kullanarak hem küresel hem de yerel değişkenleri ekrana basacak print() komutlarını tasarlayınız.

e) Fonksiyonun girintisinden çıkarak ana programa (en sola) dönünüz ve asistan_raporu_bas() yazarak hazırladığınız fonksiyonu çağırınız (tetikleyiniz).

f) Ana akışın devamına, küresel değişkene fonksiyon dışından da ulaşılabildiğini kanıtlamak için print(asistan_adi) değişkenini barındıran başarılı bir test satırı ekleyiniz.

g) Kod dosyanızın en alt satırına, fonksiyonun içinde yaşayıp ölen yerel bir değişkene dışarıdan ulaşmayı denemek için print(bellek_durumu) komutunu yorum satırı olarak ekleyiniz.

3. UYGULAMA

```
print("=== KOD-ASİSTAN: GEÇİCİ BELLEK (RAM) DENETİM PANELİ ===\n")
# KÜRESEL DEĞİŞKENLER (Global Scope)
# Bu değişkenler programın en başında tanımlandığı için tüm fonksiyonlardan okunabilir.
asistan_adi = "KODİ"
sistem_surumu = "v1.2"
def asistan_raporu_bas():
    # YEREL DEĞİŞKEN (Local Scope)
    # Bu değişken fonksiyonun içerisinde tanımlanır. Döndürülen veya başka bir yapı tarafından
referans verilen bir nesne içermiyorsa, fonksiyonun görevi bittiğinde belleği temizlenir.
    bellek_durumu = "Kararlı / RAM Optimize Edildi"

    print("[FONKSİYON İÇİ] Küresel verilere erişiliyor...")
    print(f"Çekirdek Asistan: {asistan_adi} | Sürüm: {sistem_surumu}")
    print(f"Sistem Durumu: {bellek_durumu}\n")
# ============================================================================
# ANA PROGRAM AKIŞI
# ============================================================================
print("[ANAPROGRAM] Fonksiyon çağrısı başlatılıyor...")
asistan_raporu_bas()
print("[ANAPROGRAM] Fonksiyon bitti ve fonksiyonun bellek hücresi temizlendi.")
print("-------------------------------------------------")
print(f"Ana programdan küresel veriye erişim testi: {asistan_adi} -> BAŞARILI")
# BELLEK SINIR İHLALİ TEST ALANI
# Aşağıdaki satırın başındaki (#) işaretini kaldırarak hata avcılığı yapınız.
# print(f"Ana programdan yerel veriye erişim testi: {bellek_durumu}")
```

2. Aşama: Sınır İhlali Testi ile NameError Analizi

Yazdığınız kodu terminalde çalıştırarak aşağıdaki adımları sırayla uygulayınız ve bilgisayarın ürettiği sonuçları inceleyiniz.

a) Deney (Küresel Veri Okuma): Programı ilk hâliyle çalıştırdığınızda asistan_raporu_bas() fonksiyonunun kendi içinde tanımlanmadığı hâlde asistan_adi ve sistem_surumu değişkenlerini başarıyla okuyup ekrana basabilmesini küresel etki alanı (Global Scope) kuralları çerçevesinde gerekçelendiriniz.

b) Deney (Hata Avcılığı ve Sınır İhlali): Kodun en altında yer alan bellek_durumu yazdıran print kodu satırının başındaki yorum karakterini (#) kaldırıp kodu yeniden çalıştırınız. Programın **NameError: name 'bellek_durumu' is not defined** hatasıyla aniden çökmesinin nedenini, yerel değişkenlerin bellek ömrü üzerinden nedenleriyle açıklayınız.

3. UYGULAMA

c) Deney (Gizleme-Shadowing Keşfi): asistan_raporu_bas() fonksiyonunun ilk satırına (içeriye) asistan_adi = "YENİ_KODİ" satırını ekleyiniz ve programı çalıştırınız. Fonksiyon içindeki çıktıda ismin değiştiğini ancak fonksiyon bittikten sonra ana programın altındaki çıktıda ismin hâlâ KODİ olarak kaldığını gözlemleyiniz. Python yorumlayıcısının aynı isimdeki değişkenlerde yerel veriye öncelik vererek, küresel veriyi geçici olarak gölgelemesini arkadaşlarınızla tartışınız.

3. Aşama: Projenin Dijital Depoya Yüklenmesi

a) Kararlı çalışan kodunuzu kaydedip dijital deponuza yükleyiniz.

b) Sürüm notu (Commit message) alanına "Kod-Asistan bellek yönetimi ve etki alanı modülü eklendi." yazarak deponuzu güncelleyiniz.

Değerlendirme: Çalışmanızı, karekodda yer alan Öz Değerlendirme Formu'ndaki ölçütlere göre değerlendiriniz. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Öz Değerlendirme Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

4.2.2. Kodun Fiziksel Parçalanması: Modül Oluşturma ve İçe Aktarma

Devasa bir kütüphanedeki tüm kitaplar; hiçbir kategoriye ayrılmadan, tek bir salona yığıldığında aranan bir bilgiye ulaşılması oldukça zor ve yorucu hâle gelir. Büyüyen yazılım projelerinde de yüzlerce fonksiyonun tek bir dosyanın içine sığdırılmaya çalışılması, kodların birbirine girmesine ve aranan bir hatanın bulunamamasına yol açar. Bu karmaşanın önlenmesi amacıyla birbiriyle ilişkili görevleri yerine getiren kod parçaları, kütüphanedeki farklı konu rafları (kitaplıklar) gibi fiziksel şekilde ayrı dosyalara bölünerek düzenlenir. Python dünyasında fonksiyonların ve kararlı kod bloklarının kaydedildiği bu bağımsız .py dosyalarına modül adı verilir.

Hazırlanan modüller bir diğer ifadeyle farklı dosyalarda bulunan fonksiyonlar; ana programda kullanılmak istendiğinde import (içe aktarma) komutundan yararlanılır. Bu bağlantı süreci oldukça akıcı bir şekilde gerçekleştirilir. Öncelikle yalnızca belirli bir göreve odaklanmış fonksiyonlar, bağımsız bir dosya (örneğin hesaplama_motoru.py) olarak hazırlanır. Ardından bu yeteneklerin kullanılacağı ana program dosyasının en üst satırına import hesaplama_motoru komutu yazılarak iki dosya arasında bağlantı kurulur. Son olarak dışarıdan aktarılan bir fonksiyon çalıştırılırken işlemlerin birbirine karışmasının engellenmesi ve kodun hangi kaynaktan geldiğinin açıkça belirtilmesi amacıyla modül adı ile nokta (.) operatörü birlikte kullanılır [örneğin hesaplama_motoru.indirim_yap()]. Bu fiziksel bölümlendirme sayesinde yazılan bir fonksiyonun yalnızca tek bir dosyaya hapsolmasının önüne geçilir ve fonksiyonun onlarca farklı projede defalarca yeniden kullanılabilen (reusability) bir araca dönüşmesi sağlanır.

4. UYGULAMA

Uygulamanın Adı: ÇOKLU DOSYA YÖNETİMİ VE MODÜLER ALTYAPI

Uygulamanın Süresi: 20 dakika

Uygulamanın Amacı: Bir yazılım projesini fiziksel olarak birden fazla .py dosyasına bölmek, import anahtar kelimesini kullanarak haricî bir dosyayı modül olarak ana programa dâhil etmek ve nokta (.) operatörüyle ad alanı (namespace) yönetimini ayırtetmek

Problem Senaryosu: Kod-Asistan projeniz geliştikçe tüm kodların tek bir dosyada tutulması; kodların okunmasını, yönetilmesini ve hata ayıklanmasını zorlaştırmaktadır. Bu endüstriyel karmaşayı önlemek için projenin finansal hesaplama görevlerini (örneğin indirim hesaplama) bağımsız bir dosyaya (asistan_finans.py) ayırmanız (problemi bileşenlerine ayırma) ve ardından bu yardımcı dosyayı ana programınıza (kod_asistan_ana.py) import komutuyla bağlayarak çalıştırmanız istenmektedir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Yardımcı Modülün (asistan_finans.py) Yapılandırılması

a) Kod editörü ortamında yeni bir çalışma klasörü açınız ve içerisine asistan_finans.py isimli boş bir kod dosyası oluşturunuz. Bu dosya, yalnızca finansal hesaplama fonksiyonlarını barındıran yardımcı modül olacaktır.

b) Bu dosyanın içerisine, dışarıdan sepet_tutari ve indirim_orani parametrelerini alan, indirim miktarını hesaplayıp net tutarı return ile geri fırlatan net_tutar_hesapla isimli bir fonksiyon tanımlayınız. Bu dosyada print veya input gibi ana akış komutları kullanmayınız.

2. Aşama: Ana Gövdenin (kod_asistan_ana.py) Kurulması ve Bağlantı (import)

a) Aynı klasör / dizin içerisine kod_asistan_ana.py isimli ikinci bir dosya oluşturunuz. Bu dosya, sistemin çalıştırılacağı ana program dosyası olacaktır.

b) Dosyanın en üst satırına import asistan_finans komutunu yazarak diğer dosyadaki yetenekleri ana projenize dâhil ediniz.

c) Ana akışınıza örnek sepet tutarı ve indirim oranı değişkenlerini tanımlayınız.

ç) Yardımcı modüldeki fonksiyona ulaşmak için nokta (.) operatörünü kullanarak asistan_finans.net_tutar_hesapla(...) şeklinde fonksiyonunuzu çağırınız ve elde ettiğiniz sonucu ekrana yazdırınız.

d) Ana dosyanızı terminalde çalıştırarak modüller arası veri akışının gerçekleştiğini test ediniz.

4. UYGULAMA

1. DOSYA (Yardımcı Modül Katmanı): asistan_finans.py

```
# KOD-ASİSTAN PROJESİ v1.2 - YARDIMCI FİNANSAL MODÜL
# Geliştirici: [Öğrenci Adı Soyadı]
# Standart: PEP 8 / Sadece Matematiksel Fonksiyon Hücrelerini Barındırır.
def net_tutar_hesapla(sepet_tutari, indirim_orani):
    """Gelen verileri işleyip net tutarı döndüren modül fonksiyonu"""
    indirim_miktari = sepet_tutari * (indirim_orani / 100)
    return sepet_tutari - indirim_miktari
```

2. DOSYA (Ana Yürütücü Gövde): kod_asistan_ana.py

```
# ================================================================================
# KOD-ASİSTAN PROJESİ v1.2 - MODÜLER ANA PROGRAM MOTORU
# Geliştirici: [Öğrenci Adı Soyadı]
# Standart: PEP 8 / import ve Ad Alanı (Namespace) Yönetimi
# ================================================================================
# Aynı klasördeki asistan_finans.py dosyası projeye dâhil edilir.
import asistan_finans
print("=== KOD-ASİSTAN: ÇOKLU DOSYA VE MODÜLER ALTYAPI MOTORU ===\n")
# Kullanıcı veri girişleri
girdi_sepet = 500.0
girdi_indirim = 10.0
print("[ANAPROGRAM] Harici modüldeki fonksiyon nokta (.) operatörüyle çağrılıyor...")
# AD ALANI KONTROLÜ: Fonksiyon çağrılırken hangi modülden geldiği belirtilir.
odenecek_tutar = asistan_finans.net_tutar_hesapla(girdi_sepet, girdi_indirim)
print("------------------------------------------------")
print(f"Modüler Hesaplama Sonucu Ödenecek Net Tutar: {odenecek_tutar:.2f} TL")
print("================================================")
```

3. Aşama: Sınır İhlali Testi (NameError Analizi)

Yazdığınız kodu terminalde çalıştırarak aşağıdaki deneyleri sırayla uygulayınız ve bilgisayarın ürettiği sonuçları inceleyiniz.

a) Deney (Fiziksel Ayrım Analizi): kod_asistan_ana.py dosyasını çalıştırıp çıktıyı gözlemleyiniz. Ana dosyanın içinde net_tutar_hesapla isimli bir kod satırı olmadığı hâlde işlemin hatasız yapılabilmesini import mekanizmasının donanımsal ve fiziksel rolü üzerinden açıklayınız.

4. UYGULAMA

b) Deney (Ad Alanı Sınır İhlali): Ana dosyadaki odenecek_tutar = asistan_finans.net_tutar_hesapla(...) satırını, modül ismini silerek sadece odenecek_tutar = net_tutar_hesapla(...) şeklinde değiştirip çalıştırınız. Programın neden NameError: name 'net_tutar_hesapla' is not defined hatası verdiğini nokta (.) operatörünün ve ad alanı kavramının önemini dikkate alarak yorumlayınız.

4. Aşama: Projenin Dijital Depoya Yüklenmesi

a) Birbiriyle bağlantılı çalışan bu iki dosyayı dijital deponuza yükleyiniz.

b) Sürüm notu (Commit message) alanına "Finansal fonksiyonlar ayrı bir modüle taşınarak import entegrasyonu sağlandı." yazarak deponuzu güncelleyiniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

4.2.3. Veri Akışında Esneklik: Modüllere ve Fonksiyonlara Takma İsim Vermek

Büyük ölçekli yazılım projelerinde haricî modüllerin veya modül içindeki fonksiyonların isimleri, kurumsal isimlendirme standartları gereği çok uzun olabilir. Bu uzun isimleri kod akışı içerisinde her seferinde nokta (.) operatörüyle birlikte uzun uzun yazmak, kodun okunabilirliğini azaltır ve yazım hataları riskini artırır.

Python dil mimarisinde içe aktarılan modüllere veya fonksiyonlara daha kısa, pratik ve anlamlı geçici isimler atama imkânı tanınır. Bu işleme **takma isim verme (aliasing)** adı verilir ve **as (olarak)** anahtar kelimesiyle gerçekleştirilir. Haricî bir modül, projeye dâhil edilirken import modül_adi as yeni_isim söz dizimi kullanılır. Bu işlem yapıldığında orijinal modül ismi o dosya içindeki ad alanında (namespace) askıya alınır ve Python yorumlayıcısı artık o modüle sadece atanan yeni takma isim üzerinden erişebilir. Modüllere takma isim (alias) atanmasına ilişkin temel kullanım şablonu aşağıda gösterilmiştir:

```
# Standart içe aktarma yerine takma isim (alias) kullanımı
import asistan_finans as finans
# Fonksiyon artık kısa ad alanı üzerinden çağrılır:
finans.net_tutar_hesapla(1000, 10)
```

Eğer haricî bir modülün içindeki onlarca fonksiyondan sadece tek bir tanesine ihtiyaç duyuluyorsa modülün tamamını belleğe yüklemek yerine sadece ilgili fonksiyon **from ... import ...** kalıbıyla çekilebilir. Bu fonksiyon çekilirken de "as" anahtar kelimesiyle doğrudan bir takma isim atanabilir. Bu yöntemin en büyük avantajı, çağrı esnasında artık modül adı ve nokta (.) operatörünü kullanma zorunluluğunu ortadan kaldırması, fonksiyonun doğrudan yerel bir veri gibi çağrılabilmesidir.

```
# Modülün içinden sadece tek bir fonksiyonu takma isimle çekmek
from asistan_finans import net_tutar_hesapla as net_bul
# Nokta operatörü olmadan, doğrudan takma isimle çağrı:
sonuc = net_bul(1000, 10)
```

7. ÖRNEK

MODÜL VE FONKSİYONLARDA ALIAS KULLANIMI

Kod-Asistan projenizin v1.2 sürümünde, harici .py modüllerini ve bu modüllerin içerisindeki fonksiyon hücrelerini as anahtar kelimesiyle takma isimler atayarak içe aktarmanız, ad alanı yönetimini optimize etmeniz ve kodun okunabilirliğini artırmanız istenmektedir.

Örnek Kod Bloku

1. DOSYA (Yardımcı Modül Katmanı): asistan_finans.py

```
# ============================================================================
# KOD-ASİSTAN PROJESİ v1.2 - YARDIMCI FİNANSAL MODÜL
# Standart: PEP 8 / Sadece Matematiksel Fonksiyon Hücrelerini Barındırır
# ============================================================================
def net_tutar_hesapla(sepet_tutari, indirim_orani):
    """Gelen verileri işleyip net tutarı döndüren modül fonksiyonu"""
    indirim_miktari = sepet_tutari * (indirim_orani / 100)
    return sepet_tutari - indirim_miktari
```

2. DOSYA (Ana Yürütücü Gövde): kod_asistan_ana.py

```
# ============================================================================
# KOD-ASİSTAN PROJESİ v1.2 - MODÜLER ANA PROGRAM MOTORU (ALIAS SÜRÜMÜ)
# Standart: PEP 8 / Modül ve Fonksiyon Düzeyinde Takma İsim (as) Yönetimi
# ============================================================================
print("=== KOD-ASİSTAN: TAKMA İSİM (ALIAS) VE MODÜL TEST PANELİ ===\n")
# ----------------------------------------------------------------------------
# SENARYO 1: MODÜLE TAKMA İSİM VERMEK
# asistan_finans modülü artık bu programda "fns" kısa adıyla anılacaktır.
# ----------------------------------------------------------------------------
import asistan_finans as fns
girdi_sepet_1 = 1000.0
girdi_indirim_1 = 15.0
# Çağrı yapılırken uzun modül adı yerine fns takma adı kullanılır.
```

7. ÖRNEK

```
sonuc_1 = fns.net_tutar_hesapla(girdi_sepet_1, girdi_indirim_1)
print(f"[SENARYO 1] fns Takma Adıyla Hesaplanan Tutar: {sonuc_1:.2f} TL")
# ------------------------------------------------------------------------
# SENARYO 2: FONKSİYONA DOĞRUDAN TAKMA İSİM VERMEK
# Modülden sadece ilgili fonksiyon çekilir ve "hesapla" ismiyle tescil edilir.
# ------------------------------------------------------------------------
from asistan_finans import net_tutar_hesapla as hesapla
girdi_sepet_2 = 500.0
girdi_indirim_2 = 10.0
# Nokta (.) operatörüne gerek kalmadan doğrudan çağrılır.
sonuc_2 = hesapla(girdi_sepet_2, girdi_indirim_2)
print(f"[SENARYO 2] hesapla Fonksiyon İsimlendirmesiyle Tutar: {sonuc_2:.2f} TL")
print("================================================")
```

Terminal Çıktısı

```
=== KOD-ASİSTAN: TAKMA İSİM (ALIAS) VE MODÜL TEST PANELİ ===
[SENARYO 1] fns Takma Adıyla Hesaplanan Tutar: 850.00 TL
[SENARYO 2] hesapla Fonksiyon İsimlendirmesiyle Tutar: 450.00 TL
```

Kod Analizi

Modül Düzeyinde Optimizasyon (import ... as ...): Birinci senaryoda, asistan_finans gibi uzun bir dosya adını fonksiyonu her çağırdığınızda tekrar yazmak yerine fns şeklinde kısa bir takma isim (alias) belirlenmiştir. Bu durum, kod yazım hızını artırır ve isim uzayını (namespace) daha temiz tutar.

Fonksiyon Düzeyinde Doğrudan İçe Aktarım (from ... import ... as ...): İkinci senaryoda modül yüklenmiş ancak yalnızca net_tutar_hesapla adı mevcut ad alanına aktarılmıştır. Üstelik bu fonksiyona ana programda hesapla gibi daha akılda kalıcı bir takma ad verilerek nokta (.) operatörüne duyulan ihtiyaç tamamen ortadan kaldırılmıştır.

İsim Çakışmalarını Önleme: as anahtar kelimesi, sadece yazım kolaylığı sağlamak için değil isim çakışmalarını önlemek için de kullanılır. Eğer ana programda zaten net_tutar_hesapla adında başka bir fonksiyon olsaydı dışarıdan gelen bu fonksiyona as hesapla diyerek takma ad vermek, programdaki isimlerin birbirini ezmesini ve olası mantık hatasını engellemiş olurdu.

Bilgi Kutusu

Yazılım Dünyasındaki Ünlü Takma İsimler

Profesyonel Python yazılımcıları arasında bazı küresel kütüphaneler için yaygın takma isim (alias) kullanımları bulunur. Örneğin veri bilimi yaparken yazılımcılar yaygın olarak pandas kütüphanesini import pandas as pd, NumPy'ı ise import numpy as np şeklinde kısaltarak projelerine dâhil eder. Bu sayede bir kodun içinde pd. veya np. gördüğünüzde onun hangi modüle ait olduğunu saniyeler içinde anlayabilirsiniz.

5. UYGULAMA

Uygulamanın Adı: SABİT GEÇİCİ PAROLA DÖNDÜREN MODÜLER FONKSİYON

Uygulama Süresi: 20 Dakika

Problem Senaryosu: Kod-Asistan projenizin siber güvenlik altyapısı için geçici bir sistem parolası üreten akıllı bir araç tasarlamanız istenmektedir ancak kurumsal mimari standartları gereği bu aracı ana dosyanıza yazmak yerine bağımsız bir yardımcı modülde kurgulamalısınız. Ana dosyanızdan bu modülü as anahtar kelimesiyle kısa bir takma ad vererek içe aktarmanız ve parola üretim fonksiyonunu çağırmanız gerekmektedir. Ayrıca parolayı oluştururken fonksiyon içinde kullandığınız yerel bir değişkene ana dosyadan ulaşmayı deneyerek modüller arası bellek izolasyonunun ne kadar katı olduğunu bizzat kanıtlamanız beklenmektedir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel şekilde gerçekleştiriniz.

Uygulama Adımları

a) Kod editörünüzde guvenlik_araclari.py adında yeni bir dosya oluşturunuz. İçerisine parola_uret() adında bir fonksiyon yazınız. Bu fonksiyonun gövdesine inerek (4 boşluk girintiyle) sadece orada tanımlı olan gecici_parola = "KOD-1234" şeklinde yerel bir değişken tanımlayınız ve bu değeri return komutuyla fırlatınız.

b) Aynı klasör içine ana_sistem.py adında ikinci bir dosya oluşturunuz. Yardımcı modülünüzü bir takma adla projenize bağlayınız.

c) Ana programınızda, yeni belirlediğiniz kısa ad ve nokta operatörünü kullanarak diğer dosyadaki fonksiyonu çağırınız. Dönen parolayı yeni bir değişkende yakalayıp print() ile ekrana yazdırınız. Sistemin kararlı çalıştığını test ediniz.

ç) Ana programınızın en alt satırına geçerek diğer dosyada kalmış yerel değişkene dışarıdan ulaşmayı deneyiniz. Ekranınızda beliren AttributeError veya NameError çökme mesajını analiz ederek yerel değişkenlerin dış dünyadan ne kadar izole olduğunu gözlemleyiniz.

5. UYGULAMA

d) Sınır ihlali yaptığınız hatalı satırı yorum satırı hâline getirerek programı kararlı duruma döndürünüz. Ardından her iki dosyayı da dijital deponuza yükleyerek "Güvenlik modülü alias ile entegre edildi ve modüller arası bellek izolasyonu sınır testiyle doğrulandı." sürüm notuyla dijital deponuza kaydediniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

4.3. KOD KALİTESİ VE DENETİMİ

Bir bilgisayar programının sadece hatasız çalışması, o yazılımın kaliteli ve endüstri standartlarına uygun olduğu anlamına gelmez. Yazılım mühendisliğinde **kod kalitesi (code quality)**; yazılan kodların bilgisayar sistem kaynakları üzerinde ne kadar verimli çalıştırıldığının yanı sıra başka yazılımcılar tarafından ne kadar kolay okunabildiği, anlaşılabildiği ve güncellenebildiğiyle ölçülür.

Kaliteli bir kod tabanı şu kurumsal standartlara sahip olmalıdır:

Okunabilirlik (Readability): Kod bloklarının karmaşık formüllerden uzak, temiz ve net bir söz dizimiyle yazılmasıdır.

Bakım kolaylığı (Maintainability): Sisteme yeni bir özellik eklenmek istendiğinde eski kodları yıkmadan kolayca entegre edilebilmesidir.

Yeniden Kullanılabilirlik (Reusability): Yazılan fonksiyon hücrelerinin, kopyala-yapıştır yapmaya gerek kalmadan projenin farklı alanlarında da doğrudan çalışabilmesidir.

4.3.1. Yazılım Standartları ve Teknik Dokümantasyon

Geliştirilen modüler projelerde yüzlerce farklı fonksiyon yer alabilir. Bu fonksiyonların ne iş yaptığını, dışarıdan hangi parametreleri beklediğini ve ana akışa hangi veri türünde sonuç döndürdüğünü açıklayan standart bilgi bloklarına **Docstring (Belgelendirme Dizgisi)** adı verilir.

Python dilinde Docstring'ler, **def** satırının hemen altına üçlü çift tırnak (""" ... """) arasına yazılır. Bu sayede hem projeye yeni dâhil olan yazılımcı tarafından fonksiyonun kullanım kılavuzu görülebilir hem de gelişmiş kod editörleri tarafından kod yazılırken bu açıklamalar akıllı bir ipucu olarak sunulabilir.

```
def net_tutar_hesapla(sepet_tutari, indirim_orani=10):
    """Sepet tutarına indirim uygulayarak ödenecek net tutarı hesaplar.
    Parametreler:
        sepet_tutari: İndirim uygulanmadan önceki sepet tutarıdır.
        indirim_orani: Yüzde cinsinden indirim oranıdır.
        Varsayılan değeri 10'dur.
    Döndürür:
        İndirim uygulandıktan sonra ödenecek net tutarı döndürür.
    """
    indirim_miktari = sepet_tutari * indirim_orani / 100
    net_tutar = sepet_tutari - indirim_miktari
    return net_tutar
sonuc = net_tutar_hesapla(500, 20)
```

4.3.2. Kod Yeniden Yapılandırma

Yazılım geliştirme sürecinde programın dışarıdan görünen davranışlarını, girdilerini ve çıktılarını tamamen aynı bırakarak sadece kodun iç yapısını, kalitesini ve okunabilirliğini artırmak amacıyla kaynak kodların optimize edilmesine **kod yeniden yapılandırma (refactoring)** adı verilir.

Profesyonel yazılım mühendisliğinde en çok saygı duyulan kurallardan biri **DRY [Don't Repeat Yourself (Kendini Tekrar Etme)]** prensibidir. Bir projede aynı kod blokunun ikinci kez kopyalanıp yapıştırılması, tasarım hatası yapıldığını gösterebilir. Fonksiyonlar ve modüller, kodları sadece daha okunabilir yapmakla kalmaz; aynı zamanda kod tekrarını azaltır, bakım ve güncelleme süreçlerini kolaylaştırır. Hata çıktığında yüzlerce satırı değil sadece o fonksiyonu düzeltmek yeterlidir.

Literatürde çalışan ancak gereksiz tekrarlar barındıran, değişken isimleri anlamsızlaşmış ve performansı düşüren mantıksal bozulmalara **kötü kokular (code smells)** adı verilir. Refactoring süreçlerinde yazılımcılar, kod tabanındaki kötü kokuları temizler. Örneğin yüzlerce satır süren devasa doğrusal kodlar parçalanır, mükerrer kod satırları DRY prensibine uygun olarak bağımsız fonksiyon yapılarına dönüştürülür ve karmaşık if-else merdivenleri daha yalın mantıksal süzgeçlerle sadeleştirilir. Buradaki amaç programın çalışmasını değiştirmek değil onu daha kararlı ve geleceğe hazır hâle getirmektir.



Bilgi Kutusu

Kodlarınız Spagetti mi, Lazanya mı?

Yazılım dünyasında kötü yazılmış kodların neden yemek isimleriyle anıldığını biliyor musunuz? Hiçbir modüler kurala uymayan, iç içe geçmiş ve bir hatayı ararken ipin ucunun nerede olduğunu asla bulamadığınız karmaşık kodlara küresel yazılım kültüründe **Spagetti Kod** denir. Öte yandan, katman katman ve düzenli yazılmış modüler projelere ise **Lazanya Kod** benzetmesi yapılır. Refactoring süreci, aslında o karmaşık spagetti tabağını alıp düzenli ve lezzetli bir lazanyaya çevirme sanatıdır. Siz de projelerinizi tasarlarken kodlarınızın bir spagettiye dönüşmemesine dikkat edin!

6. UYGULAMA

Uygulamanın Adı: YAPAY ZEKÂ KODLARININ KALİTE DENETİMİ, REFACTORING VE ISSUE YÖNETİMİ

Uygulamanın Süresi: 30 dakika

Uygulamanın Amacı: Yapay zekâ tarafından üretilen standart dışı ve tekrar eden kodları analiz etmek, İnsan Denetimi prensibini işleterek kod kalitesini artırmak, DRY ilkesine uygun olarak kod tabanını fonksiyonel mimariyle yeniden yapılandırmak ve bu düzeltme sürecini dijital depo üzerinden Issue takibiyle yönetmek

Problem Senaryosu: Kod-Asistan projenizin siber güvenlik ve e-ticaret katmanı için bir yapay zekâ asistanından yetkilendirme ve sepet indirim modülü talep ettiniz. Yapay zekâ ise size görünürde çalışan fakat aynı hesaplama adımlarını her kullanıcı tipi için defalarca kopyaladığı karmaşık bir kod sundu. Profesyonel bir yazılımcı olarak göreviniz; yapay zekâyı yalnızca bir asistan olarak görmek, verilen kodu adım adım süzgeçten geçirmek, gereksiz tekrarları ayıklamak ve tüm mantığı belirli kurallara göre tek bir merkezî fonksiyon altında yeniden düzenlemektir. Son olarak yaptığınız bu iyileştirmeleri dijital deponuzda bir **Görev Kartı (Issue)** ile raporlayarak kodunuzun gelişim sürecini belgelendiriniz.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Dijital Depoda Kurumsal Hata Bildirimi (Issue Açma)

a) Kodlamaya geçmeden önce internet tarayıcısından dijital deponuza girerek kod-asistan isimli proje deponuzu açınız.

b) Deponuzun üst menüsünde yer alan **Issues (Sorunlar / Görevler)** sekmesine ve ardından **New Issue (Yeni Görev)** butonuna tıklayınız.

c) Başlık (Title): "Yapay Zekâ Spagetti Kod Hatası ve DRY İhlali" yazınız.

ç) Açıklama (Body): "Yapay zekânın ürettiği yetkilendirme kodu modüler değil. Kod gereksiz tekrar içeriyor ve bakım sürecini zorlaştırıyor. Refactoring yapılarak tek bir fonksiyon çatısı altında toplanmalıdır." notunu düşerek **Submit new issue** butonuna basınız.

d) Oluşan bu görev kartının yanındaki numarayı (örneğin #1) bir yere not ediniz. Bu numara, sorunu çözerken referans olacaktır.

2. Aşama: Kod Dosyasının Hazırlanması ve Sorunlu Kodun Analizi

a) Kod editörünü açarak kod_asistan_denetim.py isimli yeni bir boş kod dosyası oluşturunuz.

b) Yapay zekâ asistanı tarafından üretilmiş olan kod blokunu satır satır inceleyiniz. Kodun aynı matematiksel ve metinsel işlemleri fonksiyon kullanmadan nasıl defalarca kopyalayıp yapıştırdığını (kod şişmesini) analiz ederek saptayınız.

6. UYGULAMA

1. DOSYA (Yapay Zekânın Ürettiği Kalitesiz Spagetti Kod)

```
# KOD-ASİSTAN PROJESİ v1.3 (STANDART DIŞI YAPAY ZEKÂ ÇIKTISI)
print("=== KOD-ASİSTAN: YAPAY ZEKÂ STANDART DIŞI KOD HAVUZU ===\n")
# YÖNETİCİ GİRİŞİ KATMANI (Tekrar Eden Blok 1)
print("[SİSTEM] Yönetici erişim katmanı doğrulanıyor...")
sepet_1 = 600
if sepet_1 > 500:
    net_1 = sepet_1 * 0.90
    print(f"Yönetici için indirim uygulandı. Ödenecek: {net_1} TL")
else:
    print(f"Yönetici için indirim yok. Ödenecek: {sepet_1} TL")
# ÖĞRETMEN GİRİŞİ KATMANI (Tekrar Eden Blok 2)
print("[SİSTEM] Öğretmen erişim katmanı doğrulanıyor...")
sepet_2 = 450
if sepet_2 > 500:
    net_2 = sepet_2 * 0.90
    print(f"Öğretmen için indirim uygulandı. Ödenecek: {net_2} TL")
else:
    print(f"Öğretmen için indirim yok. Ödenecek: {sepet_2} TL")
# ÖĞRENCİ GİRİŞİ KATMANI (Tekrar Eden Blok 3)
print("[SİSTEM] Öğrenci erişim katmanı doğrulanıyor...")
sepet_3 = 800
if sepet_3 > 500:
    net_3 = sepet_3 * 0.90
    print(f"Öğrenci için indirim uygulandı. Ödenecek: {net_3} TL")
else:
    print(f"Öğrenci için indirim yok. Ödenecek: {sepet_3} TL")
```

3. Aşama: Refactoring ve DRY Prensibi

a) Tekrar eden kod yapısını projenizden tamamen silerek yerine DRY prensibine uygun, dışarıdan kullanici_tipi ve sepet_tutari parametrelerini alan, indirimli_sistem_kontrolu() isimli tek bir merkezî fonksiyon tasarlayınız.

b) Fonksiyonunuzun hemen altına, ne işe yaradığını anlatan profesyonel bir Docstring metni eklemeyi unutmayınız.

c) Tüm if-else indirim mantığını ve hesaplamaları sadece bir kez yazarak fonksiyonun gövdesinde toplayınız.

ç) Ana akışa dönerek yazdığınız tek bir fonksiyonu farklı argümanlarla (örneğin Yönetici ve 1000 TL) çağırıp sistemin kararlı ve optimal çalıştığını test ediniz.

6. UYGULAMA

2. DOSYA (Temiz Refactoring Kodu)

```
# REFACTORING SONRASI KOD-ASİSTAN v1.3
# Geliştirici: [Öğrenci Adı Soyadı]
# Standart: PEP 8 / DRY Prensibi / Fonksiyonel Optimizasyon
print("=== KOD-ASİSTAN: REFACTORING KALİTE DENETİM MOTORU ===\n")
def indirimli_sistem_kontrolu(kullanici_tipi, sepet_tutari):
    """Kullanıcı tipini ve sepet tutarını alarak ödenecek tutarı döndürür."""
    print(f"[SİSTEM] {kullanici_tipi} erişim katmanı doğrulanıyor...")
        if sepet_tutari > 500:
        net_tutar = sepet_tutari * 0.90
        print(f"{kullanici_tipi} için indirim uygulandı. Ödenecek: {net_tutar:.2f} TL")
    else:
        net_tutar = sepet_tutari
        print(f"{kullanici_tipi} için indirim yok. Ödenecek: {sepet_tutari:.2f} TL")
            return net_tutar
# FONKSİYONUN FARKLI ARGÜMANLARLA TEKRAR KULLANILMASI (DRY UYUMU)
indirimli_sistem_kontrolu("Yönetici", 600)
indirimli_sistem_kontrolu("Öğretmen", 450)
indirimli_sistem_kontrolu("Öğrenci", 800)
```

3. Aşama: Eleştirel Kod Okuryazarlığı Deneyleri

Yeniden yapılandırılan optimize kodu terminalde yürüterek aşağıdaki adımları sırayla analiz ediniz.

a) **Deney (Verimlilik ve Kod Satırı Analizi)**: İlk kod yapısı ile refactoring uygulanmış ikinci kod yapısının terminal çıktılarını karşılaştırınız. İki kodun aynı sayısal sonuçları üretmesine rağmen ikinci kodun neden daha sürdürülebilir olduğunu açıklayınız.

b) **Deney (Hata Yönetimi Kolaylığı)**: Sisteme "İndirim sınırı 500 TL'den 700 TL'ye çıkarıldı." şeklinde yeni bir güncelleme geldiğini hayal ediniz. İlk kod yapısında bu değişikliği yapmak için kaç satıra müdahale etmeniz gerektiğini, refactoring yapılmış ikinci kodda ise sadece hangi satırı değiştirmenin yeterli olduğunu hesaplayarak fonksiyonların bakım kolaylığı üzerindeki gücünü analiz ediniz.

4. Aşama: Projenin Dijital Depoya Yüklenmesi ve Issue Kapatma

a) Refactoring işlemi tamamlanan ve kod şişmesinden kurtarılan kararlı kodunuzu dijital deponuza yükleyiniz.

b) Sürüm notu alanına "Spagetti AI kodu refactoring süzgecinden geçirilerek DRY prensibine uygun merkezî fonksiyona dönüştürüldü. **Closes #1"** (Buradaki 1 rakamı, ilk aşamada not edilen Issue numarası olmalıdır.) yazarak mesleki şeffaflık standartlarınızı koruyunuz.

6. UYGULAMA

c) Değişiklikleri kaydettiğinizde, dijital deponun Closes anahtar kelimesini algılayarak açılan Issue kartını otomatik şekilde kapattığını (Closed) ve görevin başarıyla tamamlandığını gözlemleyiniz.

Değerlendirme: Çalışmanızı, karekodda yer alan Öz Değerlendirme Formu'ndaki ölçütlere göre değerlendiriniz. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Öz Değerlendirme Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

SIRA SİZDE

1. Modüler Programlama ve Fonksiyonlar öğrenme birimi boyunca gerçekleştirdiğiniz tüm kodlama etkinliklerini, çoklu dosya yönetimlerini ve Kod-Asistan çalışmalarınızı gözünüzün önünden geçiriniz.
2. Aşağıda üç farklı geometrik sembolle tanımlanan alanlardaki (Üçgen, Kare, Çember) yönlendirme sorularını dikkatlice okuyunuz.
3. Öğrenme birimi konularıyla ilgili öğrendiklerinizden hareketle kendi öğrenme yolculuğunuzu yansıtan içten ve özgün cevaplarınızı verilen boşluklara yazınız.

ÜÇGEN: Zihnimde Şekillenen 3 Temel Kazanım

Bu öğrenme biriminin sonunda artık tam anlamıyla uzmanlaştığımı düşündüğüm, başkasına hiç zorlanmadan anlatabileceğim **3 modüler programlama beceri / bilgisi** şudur:

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

KARE: Zihnimi "Köşeye Sıkıştıran" ve Netleşmesi Gereken Pürüzler

Fonksiyon tanımlarken, parametre ve argüman eşleşmelerini kurgularken, yerel değişkenlerin gizleme (shadowing) kurallarını yorumlarken veya çoklu dosya mimarisini (import) yönetirken beni hâlâ zorlayan, zihnimde tam olarak oturmayan ve **öğretmenime sormam gereken köşeli / pürüzlü nokta** şudur:

.....

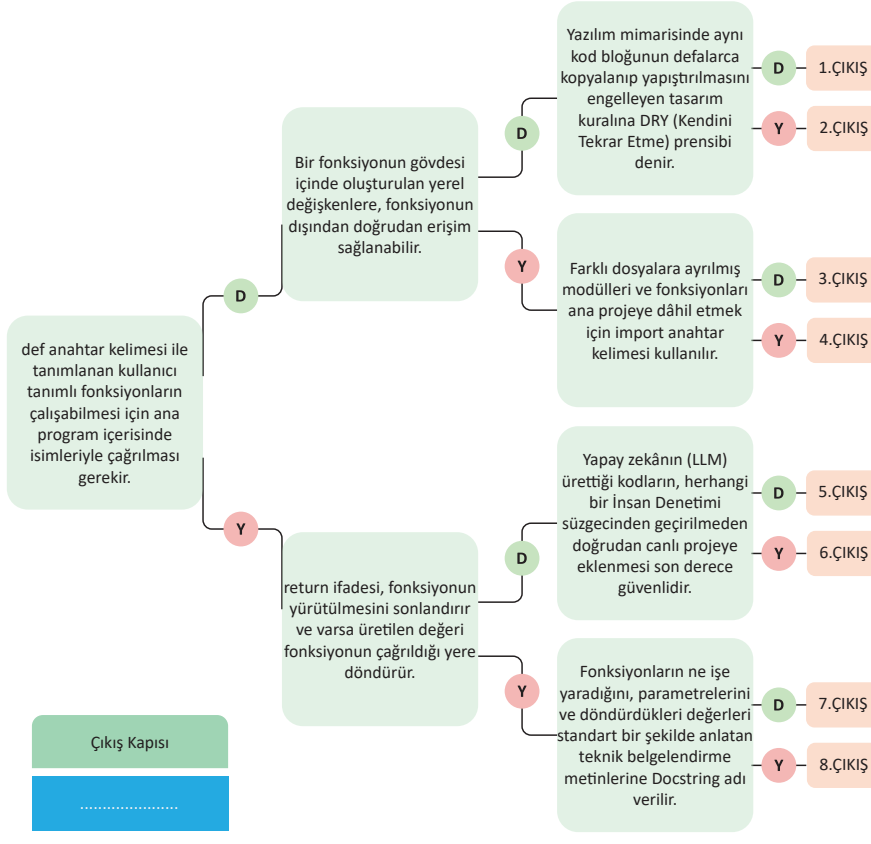
ÇEMBER: Zihnimde Dönen ve Hayatıma Değer Katan Etik Boyut

Yazılım dünyasında kod üretirken yapay zekâyı mutlak bir doğru olarak kabul etmemenin (İnsan Denetimi), çalışan ancak kalitesiz olan spagetti kodları DRY prensibiyle temizlemenin ve GitHub gibi platformlarda kurumsal Issue takibi yapmanın mesleki standartlar açısından bana kattığı ve **hayatım boyunca dikkat edeceğim temel vizyon** şudur:

.....

ÖLÇME VE DEĞERLENDİRME

A) Aşağıdaki Tanılayıcı Dallanmış Ağaç şemasında yer alan 1. ifadeden (kökten) başlayarak okuma yapınız. Okuduğunuz ifadenin doğru (D) veya yanlış (Y) olduğuna karar vererek ilgili ok yönünde (dallanarak) bir sonraki ifadeye ilerleyiniz. Her yol ayrımında verdiğiniz kararlar sizi, en son aşamada 8 farklı çıkış kapısından yalnızca birine ulaştıracaktır. Ulaşılan Çıkış Kapısı Numarasını sayfa altındaki kutucuğa işaretleyiniz.



B) Aşağıda 1'den 9'a kadar numaralandırılmış kutucuklardan oluşan yapılandırılmış grid tablosunu ve altındaki soruları dikkatlice inceleyiniz. Her sorunun cevabı veya cevapları tablodaki kutucuklarda yer almaktadır. Bir kutucuk birden fazla sorunun cevabı olarak kullanılabilir. Soruların altındaki boşluklara, doğru olan kutucuk numaralarını yazınız.

(1) def	(2) return	(3) Parametre
(4) Argüman	(5) Yerel değişken	(6) Küresel değişken
(7) import	(8) as	(9) Docstring

a) Fonksiyonların tanımlanması ve fonksiyonlar arasındaki veri akışının sağlanmasıyla doğrudan ilişkili kutucuklar hangileridir?

.....

b) Bir değişken adının programın hangi bölümlerinden erişilebilir olduğunu açıklayan etki alanı kavramları hangileridir?

.....

c) Başka bir Python dosyasındaki modül veya fonksiyonları ana programa dâhil etmek ve bunlara farklı bir ad vermek için kullanılan anahtar kelimeler hangileridir?

.....

ç) Fonksiyon çağrısı sırasında verilerin fonksiyona aktarılması ve fonksiyondan çağrıldığı yere geri gönderilmesiyle ilişkili kutucuklar hangileridir?

.....

d) Bir fonksiyonun tanımlanması, yaptığı görevin açıklanması ve ürettiği sonucun çağrıldığı yere aktarılmasıyla ilişkili kutucuklar hangileridir?

.....

e) Modüler bir programda kodun farklı dosyalarda düzenlenmesini ve bu dosyalardaki yapıların ana programda kullanılmasını sağlayan kutucuklar hangileridir?

.....

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Python dil mimarisinde özel fabrikalar (kullanıcı tanımlı fonksiyonlar) oluşturmak ve tanımlamak için kullanılan yapısal anahtar kelime aşağıdakilerden hangisidir?

A) import
B) return
C) def
D) as
E) while

2. Bir fonksiyon içinde tanımlanan yerel değişken adına fonksiyon dışından doğrudan erişilmeye çalışıldığında oluşan hata aşağıdakilerden hangisidir?

A) TypeError
B) SyntaxError
C) ValueError
D) IndexError
E) NameError

3. Fonksiyonun ürettiği veya hesapladığı bir veriyi ekrana yazdırmak yerine doğrudan çağrıldığı yere döndürmek ve aktarmak için kullanılan komut aşağıdakilerden hangisidir?

A) print
B) input
C) import
D) return
E) as

ÖLÇME VE DEĞERLENDİRME

4. Bir yazılım projesini fiziksel olarak farklı dosyalara bölüp, ardından bu harici modülleri ana program motoruna entegre etmek için kullanılan komut aşağıdakilerden hangisidir?

A) def
B) as
C) from
D) return
E) import

5. İçe aktarılan modül veya fonksiyona daha kısa bir ad vermek için kullanılan anahtar kelime hangisidir?

A) as
B) alias
C) import
D) def
E) name

6. Çalışan bir programın dış davranışını ve çıktısını kesinlikle değiştirmeden; spagetti kodları temizleyerek kodun iç mimarisini, okunabilirliğini ve modülerliğini optimize etme (iyileştirme) işlemine ne ad verilir?

A) Hata Ayıklama
B) Derleme
C) Sürümleme
D) Yeniden Yapılandırma
E) Gölgelendirme

7. Kod tekrarını azaltmayı amaçlayan "Kendini Tekrar Etme" yazılım geliştirme ilkesi hangisidir?

A) PEP 8 Standardı
B) İnsan Denetimi
C) DRY Prensibi
D) Ad Alanı Yönetimi
E) Etki Alanı Sınırı

8. Yapay zekâ tarafından üretilen bir kod blokunun doğrudan aktif sisteme dâhil edilmeden önce alanında uzman bir yazılımcı tarafından analitik bir süzgeçten geçirilmesi prensibine ne ad verilir?

A) Yapay Zekâ Halüsinasyonu
B) Modülerlik
C) Ölü Kod
D) İnsan Denetimi
E) Issue Takibi

ÖLÇME VE DEĞERLENDİRME

9. Fonksiyonların kullanım amacını, dışarıdan aldığı parametreleri ve döndürdüğü veri türlerini (return) standartlar çerçevesinde anlatan teknik dokümantasyon metinlerine ne ad verilir?

A) README
B) Docstring
C) Issue
D) Yorum Satırı (#)
E) Release

10. Aynı isme sahip bir yerel değişken ile bir küresel değişken çakıştığında, Python yorumlayıcısının fonksiyon içindeyken yerel olana öncelik vererek küresel değişkeni geçici olarak kapatması kuralına ne ad verilir?

A) Sınır İhlali
B) Ölü Kod
C) Kötü Koku
D) İçe Aktarma
E) Gizleme

Ç) Aşağıdaki soruların cevaplarını teknik terimleri kullanarak açıklayınız.

1. Aşağıdaki Python kaynak kod bloku çalıştırıldığında Python yorumlayıcısının 6. satırda programı aniden durdurduğu ve hata fırlattığı gözlemlenmiştir. Buna göre a ve b'de yer alan soruları cevaplayınız.

```
def guvenlik_duvari_olustur():
    sistem_durumu = "Güvenlik Duvarı Aktif"
    print(sistem_durumu)
print("Sistem başlatılıyor...")
guvenlik_duvari_olustur()
print(sistem_durumu)
```

a) Programın 6. satırda vereceği hatanın teknik adını yazınız.

.....

b) Hatanın nedenini, yerel değişkenin etki alanı ve fonksiyon dışından erişilebilirliği kavramlarını kullanarak açıklayınız.

.....

2. Aşağıdaki kod bloku çalıştırıldığında TypeError vererek sonlanmaktadır. Buna göre a ve b'de yer alan soruları cevaplayınız.

```
def indirim_hesapla(tutar):
    indirim = tutar * 0.10
    print(indirim)
# Ana Program
sepet_tutari = 500
odenecek_net_tutar = sepet_tutari - indirim_hesapla(sepet_tutari)
```

ÖLÇME VE DEĞERLENDİRME

a) print() ve return komutları arasındaki mimari fark nedir? Ana programdaki matematiksel çıkarma işleminin neden hata verdiğini veri aktarımı bağlamında açıklayınız.

……
……

b) Bir fonksiyonun içinde return komutundan hemen alt satıra yazılan ve Python yorumlayıcısının tarafından hiçbir zaman ulaşılamayan kod satırlarına literatürde ne ad verilir?

……
……

3. Yazılım projenizdeki matematiksel işlemleri asistan_finans.py isimli harici bir dosyaya kaydettiniz. Ana programınızda bu dosyayı içe aktarırken çok uzun olduğu için import asistan_finans as fns komutunu kullandınız. Buna göre a ve b'de yer alan soruları cevaplayınız.

a) İçe aktarılan bu harici modüle fns adını atamanızı sağlayan as anahtar kelimesinin kullanım işlemine literatürde ne ad verilir?

……
……

b) asistan_finans.py modülünün içinde yer alan kdv_ekle(tutar) isimli fonksiyonu ana programda çağırmak için yazmanız gereken doğru kod satırını, yeni kısa adınızı ve nokta (.) operatörünü kullanarak yazınız.

……
……

4. Bir yapay zekâ asistanından sepet tutarını hesaplayan bir fonksiyon yazmasını istediniz. Yapay zekâ size saniyeler içinde çalışan bir fonksiyon üretti ancak ürettiği fonksiyonun ne işe yaradığını anlatan bir teknik dokümantasyonu yok ve değişken isimlerini Türkçe karakter kullanarak sepettutarı şeklinde yazmış. Bir bilişim uzmanı olarak bu kodu aktif sisteme doğrudan kopyalamak yerine İnsan Denetimi prensibi çerçevesinde nasıl bir yaklaşım sergilemeniz gerektiğini ve yapay zekâyı projenizde nasıl konumlandırmanız gerektiğini etik ve teknik gerekçeleriyle açıklayınız.

……
……

5. GitHub gibi bir dijital depolama platformunda, yapay zekânın ürettiği spagetti kodları düzeltmek için ekibinizle ortaklaşa çalışıyorsunuz. Buna göre a ve b'de yer alan soruları cevaplayınız.

a) Dijital depoda bir sorunu raporlamak ve düzeltme görevini resmî olarak başlatmak için oluşturduğunuz kayıt kartına ne ad verilir?

……

b) Kodları temizleyip dijital deponuza yüklerken oluşturduğunuz bu görev kartını otomatik olarak kapatmak ve görevi tamamladığınızı sisteme bildirmek için sürüm notuna yazmanız gereken akıllı etiket (örneğin 1 numaralı görev için) nedir?

……



Öğrenme Birimi Karekodu

5. ÖĞRENME BİRİMİ
VERİ YÖNETİMİ VE KALICI PROGRAM YAPILARI

KONULAR

5.1. GELİŞMİŞ VERİ YAPILARI (DICTIONARY)
5.2. DOSYA İŞLEMLERİ (I/O)
5.3. VERİ FORMATLARI VE DENETİM

ANAHTAR KAVRAMLAR

- anahtar-değer (key-value)
- dosya işlemleri
- JSON
- kalıcılık (persistence)
- sözlük (dictionary)
- TXT
- with open() kalıbı

BEKLENTİLER

Bu öğrenme biriminde sizden

- veri yapılarını (liste ve sözlük), verileri gruplandırmak için kullanmanız
- dosya sistemlerini kullanarak verileri kalıcı şekilde saklamanız
- yapay zekâ desteğiyle veri yönetimini tasarlayarak iyileştirmeniz

beklenmektedir.

BAŞLARKEN

On binlerce kitabın bulunduğu devasa bir kütüphanede aradığınız tek bir cümleyi saniyeler içinde bulabildiğinizi hayal edin. Peki, bilgisayarlar bu kusursuz hafıza yönetimini ve bilgiye nokta atışı erişimi dijital dünyada nasıl sağlar?

Bugüne kadar komutları yerine getiren Kod-Asistan, her kapandığında hafızasını kaybediyor ve eski bilgileri unutuyordu. Bu öğrenme biriminde asistan nihayet kalıcı bir dijital hafızaya kavuşuyor! Bilgileri bir kütüphane titizliğiyle organize eden ve bilgisayarın fişi çekilse bile hiçbir şeyi unutmayan gerçek bir dijital yardımcı tasarlamaya hazır mısınız?

HAZIRLIK ÇALIŞMALARI

1. Bir okul kütüphanesindeki binlerce kitabın herhangi bir kayıt veya sınıflandırma sistemi olmadan raflara gelişigüzel yerleştirildiğini düşününüz. Aradığınız belirli bir kitaba ulaşmaya çalışırken yaşayacağınız süreci ve bu durumun olası sonuçlarını arkadaşlarınızla tartışınız.
2. Verilen anahtar kavramın size neleri çağrıştırdığını düşününüz. Bu kavramla ilgili aklınıza gelen en fazla 5 kelimeyi ilgili kutucuklara yazınız. Ardından bu kelimeleri kullanarak kavram hakkındaki ön bilgilerinizi yansıtan anlamlı bir cümle kurunuz.

Anahtar Kavram: DİJİTAL HAFIZA (KAYIT)

a)	ç)
b)	d)
c)	

İlgili Cümlem:

.....

.....

5.1. SÖZLÜK VERİ YAPISI (DICTIONARY)

Listeler, verileri belirli bir sıraya göre saklayan ve bu verilere indeks numaraları (0, 1, 2, ...) aracılığıyla erişilmesini sağlayan veri yapılarıdır. Bununla birlikte gerçek yaşam problemlerinde verilere çoğu zaman sıralarına göre değil taşıdıkları anlamlara göre erişmek gerekir. Örneğin bir öğrenciye ait bilgiler saklanırken öğrenci numarası, adı, sınıfı, not ortalaması gibi farklı türde veriler birbiriyle ilişkilidir. Bu verilere ulaşmak için sürekli indeks numaralarını takip etmek hem okunabilirliği azaltır hem de veri miktarı arttıkça yönetimi zorlaştırır.

Bir okul kütüphanesinde kitaplar aranırken kitabın raftaki sırasından çok kitap adı, barkod numarası veya yazar bilgisi kullanılır. Benzer şekilde bir telefon rehberinde de kişilere sıra numarasıyla değil isimleriyle ulaşılır. Bu tür durumlarda verileri anlamlı etiketlerle ilişkilendirmek daha kullanışlı bir yaklaşım sunar.

Python programlama dilinde verileri anlamlı etiketler kullanarak saklamayı sağlayan veri yapısına **sözlük (dictionary)** adı verilir. Sözlükler, verileri **anahtar-değer (key-value)** ilişkisiyle organize eder ve bir bilgiye doğrudan ilgili anahtar üzerinden erişilmesini sağlar. Sözlük yapısındaki her bilgi iki bölümden oluşur.

Anahtar (Key): Veriye erişmek için kullanılan benzersiz etikettir. Sözlük içerisinde aynı anahtar birden fazla kez kullanılamaz.

Değer (Value): Anahtar ile ilişkilendirilen bilgidir. Metin, sayı, liste veya başka bir sözlük olabilir.



Bir kütüphane fihristinde barkod numarasının kitabı temsil etmesi ya da telefon rehberinde kişinin adının telefon numarasıyla ilişkilendirilmesi, anahtar-değer ilişkisine örnek olarak gösterilebilir. Python'da sözlükler, süslü parantezler **{ }** kullanılarak tanımlanır. Her anahtar ile değeri, iki nokta **(:)** işareti birbirine bağlar.

Örnek Kod Bloku

```
# Standart bir sözlük tanımlama örneği
ogrenci = {
   "numara": 125,
   "ad": "Ayşegül",
   "sinif": "9-A",
   "ortalama": 88.5
}
```

Kod Analizi

- "numara", "ad", "sinif" ve "ortalama" ifadeleri anahtardır.
- 125, "Ayşegül", "9-A" ve 88.5 değerlerdir.

Bu yapı sayesinde veriler yalnızca sıralı olarak değil anlamlı ilişkiler kurularak da saklanabilir. Böylece programlar daha okunabilir, daha düzenli ve yönetilebilir hâle gelir.

5.1.1. Sözlük Elemanlarına Erişim ve Güvenli Veri Erişimi

Sözlüklerde bir değere ulaşmak için ilgili anahtar köşeli parantez ([]) içerisinde belirtilir. Listelerde verilere indeks numaralarıyla erişilirken sözlüklerde anlamlı anahtarlar kullanılır. Bu sayede verilerin neyi temsil ettiği daha kolay anlaşılır.

Örnek Kod Bloku

```
print("Öğrencinin Adı:", ogrenci["ad"])
print("Öğrencinin Sınıfı:", ogrenci["sinif"])
# Sözlükte bulunmayan bir anahtarı çağırmak, bir KeyError hatasına yol açar!
# print(ogrenci["soyad"])
```

Kod Analizi

1. Adım (Anlamlı ve Doğrudan Erişim): Sözlük veri yapısında değerlere listelerdeki gibi sayısal indeks numaralarıyla değil "ad", "sinif" gibi veriyi tanımlayan anlamlı anahtarlarla erişilir. Bu yapı, Python yorumlayıcısının doğrudan hedef veriye ulaşmasını sağlarken kodun okunabilirliğini de büyük ölçüde artırır.

2. Adım [Çalışma Zamanı Hatası (KeyError)]: Sözlükte tanımlanmamış bir anahtara (örneğin "soyad") köşeli parantez [] kullanılarak doğrudan erişilmeye çalışıldığında Python yorumlayıcısı bu etiketi bulamaz ve KeyError (Anahtar Hatası) vererek programın akışını aniden sonlandırır.

Bir sözlükte bulunmayan bir anahtara doğrudan erişilmeye çalışıldığında Python KeyError hatası üretir.

Bu durum, programın beklenmeyen şekilde sonlanmasına neden olabilir. Bir anahtarın sözlükte bulunup bulunmadığından emin olunmadığında **.get()** metodu kullanılabilir. Bu metot, belirtilen anahtar sözlükte varsa ilgili değeri döndürür. Anahtar bulunmuyorsa hata oluşturmak yerine None değerini veya kullanıcı tarafından belirlenen varsayılan bir değeri döndürür. .get() metodu, sözlükte bulunmayan bir anahtara erişilmeye çalışıldığında programın hata vermesini önlemeye yardımcı olur. Bu nedenle sözlüklerdeki veri sorgulama işlemlerinde sıkça kullanılır.

Örnek Kod Bloku

```
# Güvenli veri erişimi örneği
kitap_bilgisi = {
   "ad": "Dîvân-ı Hikmet",
   "yazar": "Hoca Ahmed Yesevî"
}
print(f"Kitap Adı: {kitap_bilgisi.get('ad', 'Kitap adı bilgisi bulunamadı.')}")
# Anahtar bulunmuyorsa varsayılan değer döndürülür.
yayinevi = kitap_bilgisi.get("yayinevi", "Yayınevi bilgisi bulunamadı.")
print(f"Yayınevi: {yayinevi}")
# Var olan bir anahtarın değeri alınır.
yazar = kitap_bilgisi.get("yazar", "Yazar bilgisi bulunamadı.")
print(f"Yazar: {yazar}")
```

Terminal Çıktısı

```
Kitap Adı: Dîvân-ı Hikmet
Yayınevi: Yayınevi bilgisi bulunamadı.
Yazar: Hoca Ahmed Yesevî
```

Kod Analizi

1. Adım [Güvenli Veri Sorgulama (KeyError Önleme)]: Sözlükte tanımlanmamış olan "yayinevi" anahtarına köşeli parantez [] ile doğrudan erişilmeye çalışılsaydı Python yorumlayıcısı KeyError hatası vererek programı aniden sonlandıracaktı ancak .get() metodu kullanılarak bu hata engellenmiş ve yazılımın kararlılığı (stabilitesi) sağlamıştır.

2. Adım (Dinamik Varsayılan Değer Üretimi): .get() metodu, bulunamayan "yayinevi" anahtarı için programın akışını bozmak yerine, ikinci parametre olarak tanımlanan "Yayınevi bilgisi bulunamadı." metnini varsayılan değer (fallback) olarak sisteme döndürerek hata yönetimi sürecini hatasız işletmiştir.

3. Adım (Mevcut Verilere Kesintisiz Erişim): Sözlükte zaten varlığından emin olunan "ad", "yazar" gibi anahtarlar çağrıldığındaysa .get() metodu doğrudan bellekteki orijinal verileri (Dîvân-ı Hikmet ve Hoca Ahmed Yesevî) başarıyla ana akışa aktarmıştır.



5.1.2. Sözlüklerde Veri Ekleme ve Güncelleme

Programlar çalışırken yeni bilgiler eklenebilir veya mevcut bilgiler değiştirilebilir. Sözlük yapıları da bu tür değişikliklerin kolayca gerçekleştirilmesine olanak sağlar. Bir sözlüğe yeni bir anahtar-değer çifti eklemek veya mevcut bir değeri güncellemek için köşeli parantez içerisinde ilgili anahtar belirtilir ve yeni değer atanır. Bir sözlükte bulunmayan bir anahtara değer atanırsa yeni bir kayıt oluşturulur. Anahtar zaten mevcutsa eski değer yeni değer ile değiştirilir.

Örnek Kod Bloku

```
kitap_bilgisi = {
   "ad": "Dîvân-ı Hikmet",
   "yazar": "Hoca Ahmed Yesevî"
}
# Sözlüğe daha önce bulunmayan ‘tür’ anahtarı ve değeri eklenir.
kitap_bilgisi["tür"] = "Tasavvuf"

print(kitap_bilgisi)
```

Terminal Çıktısı

{'ad': 'Dîvân-ı Hikmet', 'yazar': 'Hoca Ahmed Yesevî', 'tür': 'Tasavvuf'}

Kod Analizi

1. Adım (Dinamik Bellek Genişlemesi): Python sözlükleri, programın çalışma zamanında boyutları esnek bir şekilde büyüyebilen dinamik veri yapılarıdır. kitap_bilgisi["tür"] = "Tasavvuf" komutu çalıştırıldığında Python yorumlayıcısı "tür" anahtarının bellekte daha önce tanımlanmadığını saptar, hata vermek yerine yeni anahtar-değer çifti sözlüğe eklenir.

2. Adım (Doğrudan Atama ile Pratik Entegrasyon): Listelere yeni bir veri eklemek için özel yerleşik metotlara (örneğin .append()) ihtiyaç duyulurken, sözlük mimarisinde doğrudan köşeli parantez [] ve atama operatörü = kullanılarak yeni bir anahtar-değer çifti kod tabanına son derece hızlı, pratik ve düşük maliyetli bir şekilde entegre edilmiştir.

Örnek Kod Bloku

```
kitap_bilgisi = {
   "ad": "Dîvân-ı Hikmet",
   "yazar": "Hoca Ahmed Yesevî",
   # Bu şekilde aynı isimde  birden fazla anahtar kullanılması mantık hatasıdır.
   "tür": "Diğer",
   "tür": "Tasavvuf"
}
# Var olan 'yazar' anahtarının değeri güncellenir
kitap_bilgisi["yazar"] = "Pir-i Türkistan Hoca Ahmed Yesevî"
print(kitap_bilgisi)
```

Terminal Çıktısı

{'ad': 'Dîvân-ı Hikmet', 'yazar': 'Pir-i Türkistan Hoca Ahmed Yesevî', 'tür': 'Tasavvuf'}

Kod Analizi

1. Adım [Değerin Üzerine Yazma (Overwriting)]: Python sözlüklerinde her bir anahtar benzersiz (unique) olmak zorundadır. Atama operatörü = kullanılarak sözlükte zaten var olan bir anahtara ("yazar") yeni bir veri gönderildiğinde Python yorumlayıcısı eski veriyi (Hoca Ahmed Yesevî) yeni veriyle (Pir-i Türkistan Hoca Ahmed Yesevî) değiştirir.

2. Adım [Esnek Veri Manipülasyonu (Mutability)]: Sözlükler çalışma sırasında değiştirilebilir veri yapılarıdır. Bu nedenle sözlüğü tamamen yeniden oluşturmaya gerek kalmadan belirli bir anahtarın değeri doğrudan değiştirilebilir.

5.1.3. Sözlüklerden Veri Silme

Yazılım projelerinin çalışma sürecinde bazı verilere artık ihtiyaç duyulmayabilir veya bellek yönetimini optimize etmek amacıyla belirli kayıtların sistemden tamamen temizlenmesi gerekebilir. Bir sözlükteki belirli bir anahtarı ve o anahtara bağlı olan değeri kalıcı olarak kaldırmak için veri silme komutları; sözlüğün genel mimarisini, etiketlerini veya taşıdığı tüm verileri topluca analiz etmek için ise içeriği görüntüleme metotları kullanılır.

Bir sözlük içerisindeki belirli bir elemanı sözlükten kaldırmak için **del (delete)** anahtar kelimesi kullanılır. Bu işlem gerçekleştirilirken silinmek istenen verinin anahtarı köşeli parantez içinde belirtilir.

Örnek Kod Bloku

```
kitap_bilgisi = {
   "ad": "Dîvân-ı Hikmet",
   "yazar": "Hoca Ahmed Yesevî",
   "tür": "Tasavvuf"
}
# İhtiyaç duyulmayan ‘tür’ anahtarı ve değeri sözlükten silinir
del kitap_bilgisi["tür"]
print(kitap_bilgisi)
```

Terminal Çıktısı

{‘ad’: ‘Dîvân-ı Hikmet’, ‘yazar’: ‘Hoca Ahmed Yesevî’}

Kod Analizi

1. Adım (Silme): Bu işlem sonucunda "tür" anahtarı ve ona bağlı olan "Tasavvuf" değeri sözlükten silinir.

2. Adım (KeyError Riski): Silme işlemi yapılırken, sözlükte o an var olmayan bir anahtarı (örneğin daha önce silinmiş olan "tür" anahtarını ikinci kez) silmeye çalışmanın KeyError üreterek hata vereceği unutulmamalıdır. Yazılımın kararlılığını korumak için silme işleminden önce anahtarın varlığı denetlenebilir.



Sözlükten veri silerken del komutunun yanı sıra silinen veriyi kaybetmeden önce yedeklemek veya sözlüğün iskeletini bozmadan içini tek seferde tamamen boşaltmak için Python'un yerleşik metotlarından yararlanılabilir.

Örnek Kod Bloku

```
kitap_bilgisi = {
   "ad": "Dîvân-ı Hikmet",
   "yazar": "Hoca Ahmed Yesevî",
   "tür": "Tasavvuf",
   "sayfa": 144
}
# .pop() metodu ile anahtarı silme ve değerini yedekleyerek yakalama
silinen_veri = kitap_bilgisi.pop("tür")
print(f"[BİLGİ] '{silinen_veri}' bilgisi bellekten silindi.")
# .clear() metodu ile sözlüğün içindeki tüm kayıtları sıfırlama
kitap_bilgisi.clear()
print(f"Sözlüğün Son Durumu: {kitap_bilgisi}")
```

Terminal Çıktısı

```
[BİLGİ] 'Tasavvuf' bilgisi bellekten silindi.
Sözlüğün Son Durumu: {}
```

Kod Analizi

1. Adım [Değer Döndürerek Güvenli Silme (.pop())]: del komutu veriyi sözlükten kaldırır ancak .pop() metodu, silinen anahtara ait değeri sözlükten silmeden hemen önce programa geri döndürür (return). Bu sayede veri sözlükten silinirken aynı zamanda silinen_veri adlı değişkende tutularak işlem geçmişine (log) yazılabilmiş veya son bir kez değerlendirilebilmiştir.

2. Adım [Tamamen Sıfırlama (.clear())]: del kitap_bilgisi komutu değişken adı ile sözlük bağlantısını kesip NameError oluşmasına yol açarken .clear() metodu sözlüğün ismini korur. Sadece içerisindeki tüm verileri temizleyerek geriye yeniden veri eklenmeye hazır boş bir sözlük {} bırakır.

5.1.4. Sözlük İçeriğini Toplu Görüntüleme ve Analiz Metotları

Karmaşık veri kümeleriyle çalışırken sözlüğün içindeki tüm anahtarları, tüm değerleri veya her ikisini birden topluca görmek ve özellikle for döngüleriyle işlemek için Python'un yerleşik sözlük metotlarından yararlanılır.

Örnek Kod Bloku

```
kitap_bilgisi = {
   "ad": "Dîvân-ı Hikmet",
   "yazar": "Hoca Ahmed Yesevî"
}
print("Anahtarlar:", kitap_bilgisi.keys())
print("Değerler:", kitap_bilgisi.values())
print("Anahtar-Değer Çiftleri:", kitap_bilgisi.items())
```

Örnek Çıktı

Anahtarlar: dict_keys(['ad', 'yazar'])
Değerler: dict_values(['Dîvân-ı Hikmet', 'Hoca Ahmed Yesevî'])
Anahtar-Değer Çiftleri: dict_items([('ad', 'Dîvân-ı Hikmet'), ('yazar', 'Hoca Ahmed Yesevî')])

Kod Analizi

1. Adım (keys() Metodu): Sözlükte bulunan anahtarları liste benzeri bir görünüm nesnesi olarak döndürür. Sözlükte hangi anahtarların bulunduğunu görüntülemek veya bu anahtarlar üzerinde işlem yapmak için kullanılır.

2. Adım (values() Metodu): Sözlükteki değerleri liste benzeri bir görünüm nesnesi olarak döndürür. Örneğin sözlükte kayıtlı eser adlarının veya yazar bilgilerinin yalnızca değerlerini görüntülemek ve işlemek için kullanılabilir.

3. Adım (items() Metodu): Sözlükteki her bir anahtar-değer çiftini ikili yapılar hâlinde içeren bir görünüm nesnesi döndürür. Sözlükteki anahtar ve değerlerin birlikte incelenmesi ve özellikle bir for döngüsüyle her bir veri çiftinin sırayla işlenmesi için kullanılır.

1. ÖRNEK

ANAHTAR-DEĞER MANTIĞI VE ENVANTER YÖNETİMİ

Bir e-ticaret sisteminde ürünlerin stok bilgileri takip edilmektedir. Ürün isimleri ile stok miktarlarının ilişkilendirilerek saklanması, verilere daha düzenli ve anlaşılır biçimde erişilmesini sağlar. Bu kapsamlı senaryoda sözlük oluşturma, anahtar kullanarak değere ulaşma, yeni veri ekleme, mevcut veriyi değiştirme (güncelleme), gereksiz veriyi silme (del) işlemleri ile sözlük içeriğini analiz eden gelişmiş metotların (keys, values, items) bir for döngüsüyle harmanlanması test edilecektir.

Örnek Kod Bloku

```
print("--- DİJİTAL ENVANTER YÖNETİM SİSTEMİ ---")
# 1. Aşama: Sözlük oluşturma
stok_envanteri = {
    "Laptop": 45,
    "Telefon": 120,
    "Kulaklık": 300
}
# 2. Aşama: Anahtar kullanarak değere erişim
telefon_stogu = stok_envanteri["Telefon"]
print(f"Güncel Telefon Stoğu: {telefon_stogu} adet")

# 3. Aşama: Yeni veri ekleme
stok_envanteri["Tablet"] = 50
```



1. ÖRNEK

```
print("\n[BİLGİ] Tablet ürünü envantere eklendi.")
# 4. Aşama: Mevcut değeri değiştirme (Güncelleme)
stok_envanteri["Laptop"] = 40
print("[BİLGİ] Laptop stoğu 40 olarak güncellendi.")
# 5. Aşama: Sözlükten veri silme (del)
del stok_envanteri["Kulaklık"]
print("[BİLGİ] Kulaklık ürünü envanterden tamamen silindi.")
# 6. Aşama: Sözlük metotları ile içeriği görüntüleme (keys, values, items)
print("\n--- ENVANTER METOT ANALİZİ ---")
print("Anahtarlar (Kayıtlı Ürünler):", stok_envanteri.keys())
print("Değerler (Stok Miktarları):", stok_envanteri.values())
print("Çiftler (Tüm Kayıtlar):", stok_envanteri.items())
# 7. Aşama: for döngüsü kullanımı ile listeleme
print("\n--- GÜNCEL ENVANTER DURUMU (FOR DÖNGÜSÜ İLE) ---")
for urun, adet in stok_envanteri.items():
    print(f"-> {urun} : {adet} adet stokta mevcuttur.")
```

Terminal Çıktısı

--- DİJİTAL ENVANTER YÖNETİM SİSTEMİ ---

Güncel Telefon Stoğu: 120 adet

[BİLGİ] Tablet ürünü envantere eklendi.

[BİLGİ] Laptop stoğu 40 olarak güncellendi.

[BİLGİ] Kulaklık ürünü envanterden tamamen silindi.

--- ENVANTER METOT ANALİZİ ---

Anahtarlar (Kayıtlı Ürünler): dict_keys(['Laptop', 'Telefon', 'Tablet'])

Değerler (Stok Miktarları): dict_values([40, 120, 50])

Çiftler (Tüm Kayıtlar): dict_items([('Laptop', 40), ('Telefon', 120), ('Tablet', 50)])

--- GÜNCEL ENVANTER DURUMU (FOR DÖNGÜSÜ İLE) ---

-> Laptop : 40 adet stokta mevcuttur.

-> Telefon : 120 adet stokta mevcuttur.

-> Tablet : 50 adet stokta mevcuttur.

1. ÖRNEK

Kod Analizi

1. Adım (Oluşturma ve Değer Atama): Sözlük yapısında veriler, anahtar-değer ilişkisiyle bir çatı altında toplanmış ve saklanmıştır. Belirli bir veriye ulaşmak için indeksler yerine anlamlı anahtarlar ("Telefon") kullanılarak doğrudan erişim hızı sağlanmıştır.

2. Adım [Dinamik Ekleme ve Değiştirme (Mutability)]: Sözlükler çalışma anında güncellenebilen yapılardır. stok_envanteri["Tablet"] = 50 komutuyla bellekte olmayan yeni bir ürün sözlüğe eklenirken, aynı işlem mevcut olan "Laptop" için yapıldığında, eski değerin üzerine yazılarak veri anlık olarak değiştirilmiştir.

3. Adım [Tamamen Silme (del)]: Sistemde artık ihtiyaç duyulmayan "Kulaklık" kaydı del komutu kullanılarak sözlükten kalıcı olarak temizlenmiştir.

4. Adım [Toplu Analiz (keys, values, items)]: Sözlüğe ait yerleşik metotlar kullanılarak sistemde kayıtlı ürün adları ve stok değerleri ayrı ayrı listelenmiş ve incelenmiştir.

5. Adım (for Döngüsü Entegrasyonu): Kurumsal bir yazılımın en çok ihtiyaç duyduğu işlemlerden biri olan **veri raporlama**, sözlüğün items() metoduyla bir for döngüsüne sokulmasıyla gerçekleştirilmiştir. Bu sayede sözlükteki tüm kayıtlar tek tek işlenerek son kullanıcıya temiz ve okunabilir bir formatta sunulmuştur.

5.1.5. İç İçe Yapılarla Veri Modelleme: Listeler ve Sözlükler

Gerçek yaşam verileri çoğu zaman tek bir liste veya tek bir sözlük içerisinde tutulamayacak kadar ayrıntılıdır. Bir eserin adı, yazarı, türü gibi bilgiler birbirleriyle ilişkilidir. Benzer şekilde bir yazarın birden fazla eseri veya bir eserin birden fazla türü olabilir. Bu tür durumlarda listeler ve sözlükler birlikte kullanılarak daha düzenli veri modelleri oluşturulur.

Bir listenin içerisinde sözlükler, bir sözlüğün içerisinde ise listeler veya başka sözlükler bulunabilir. Bu yapılara **iç içe veri yapıları (nested structures)** adı verilir.

- Bir kütüphanede yüzlerce eser bulunabilir ve her eserin adı, yazarı, türü gibi bilgileri vardır. Bu bilgileri düzenli şekilde saklamak için liste içinde sözlük kullanımı tercih edilebilir; yani her eser bir sözlük olarak tanımlanabilir ve tüm eserler bir liste içerisinde tutulabilir. Verilen örnekte kütüphanedeki eserler bir liste içerisinde saklanmaktadır ve listenin her elemanı bir sözlüktür.



Örnek Kod Bloku

```
eserler = [
  {
    "eser": "Dîvân-ı Hikmet",
    "yazar": "Hoca Ahmed Yesevî",
    "tur": "Tasavvuf"
  },
  {
    "eser": "Mesnevî",
    "yazar": "Mevlânâ",
    "tur": "Tasavvuf"
  },
  {
    "eser": "Risaletü'n Nushiyye",
    "yazar": "Yunus Emre",
    "tur": "Öğüt"
  }
]
print(eserler[1]["eser"])
```

Terminal Çıktısı

Mesnevî

Kod Analizi

1. Adım [Veri Hiyerarşisi (Liste İçinde Sözlük)]: Bu mimaride eserler köşeli parantez [] ile tanımlanmış bir listedir. Listenin içerisindeki her bir eleman ise süslü parantez {} ile kurgulanmış bağımsız birer sözlüktür. Bu sayede her bir sözlük, tek bir esere ait bilgileri (eser adı, yazar ve tür) anahtar-değer ilişkisiyle kendi içinde bütünlük sağlayarak saklamaktadır.

2. Adım (Zincirleme Erişim): Karmaşık verilerde hedef değere ulaşmak için zincirleme bir adresleme yöntemi kullanılmıştır. Önce eserler komutuyla listenin ikinci elemanına (Mevlânâ'nın eseri) gidilmiş, ardından bu indeksin içindeki sözlüğe ait olan "eser" anahtarı ["eser"] şeklinde yanına eklenerek Mesnevî çıktısı elde edilmiştir.

Örnek Kod Bloku

```
yazar = {
  "ad": "Yunus Emre",
  "eserler": [
    "Risaletü'n Nushiyye",
    "Divan"
  ]
}
# "eserler" anahtarına bağlı olan listeyi ekrana yazdırma
print(yazar["eserler"])
```

Terminal Çıktısı

['Risaletü'n Nushiyye', 'Divan']

Kod Analizi

1. Adım (Çoklu Veri Gruplama): Bu mimaride "eserler" anahtarının karşılığına doğrusal bir liste yerleştirilmiştir. Böylece bir yazara ait birden fazla eser, yeni anahtarlar oluşturmadan tek bir etiket altında gruplandırılmıştır. İç içe yapılarda ve büyük veri setlerinde, aranan bazı alt anahtarlar her kayıtta her zaman bulunmayabilir. Bu belirsizlik durumlarında programın hata vermesini engellemek ve veriye güvenli erişim sağlamak için **.get() metodu** kullanılır.

Örnek Kod Bloku

```
yazar = {
    "ad": "Hoca Ahmed Yesevî",
    "kisisel_bilgiler": {
        "unvan": "Pir-i Türkistan"
        # 'dogum_yeri' anahtarı bu alt sözlükte tanımlanmamıştır.
    }
}
# İç içe yapıdaki alt sözlüğe girerek bulunmayan bir anahtarı güvenli şekilde
sorgulama
dogum_yeri = yazar["kisisel_bilgiler"].get("dogum_yeri", "Bilgi bulunamadı.")
print(dogum_yeri)
```

Terminal Çıktısı

Bilgi bulunamadı.

Kod Analizi

1. Adım (Hiyerarşik ve Güvenli Sorgulama): İlk adımda yazar sözlüğünün içindeki "kisisel_bilgiler" alt sözlüğüne köşeli parantez [] ile inilmiş ancak bu alt sözlüğün içinde "dogum_yeri" anahtarının varlığından emin olunamadığı için ikinci adımda .get() metodu devreye sokulmuştur.

2. Adım [Çalışma Zamanı (Runtime) Koruması]: Hedeflenen "dogum_yeri" anahtarı alt sözlükte tanımlı olmadığı için zincirleme olarak yazar["kisisel_bilgiler"]["dogum_yeri"] şeklinde doğrudan çağrılsaydı program KeyError (Anahtar Hatası) verecekti. Alt sözlüğe eklenen .get() metodu sayesinde sistemin çökmesi engellenmiş ve eksik veri durumunda akışı bozmayacak varsayılan bir bilgi mesajı üretilmiştir.

1. UYGULAMA

Uygulamanın Adı: DİJİTAL ESER KATALOĞU VE İÇ İÇE SÖZLÜK MİMARİSİ

Uygulamanın Süresi: 30 dakika

Uygulamanın Amacı: Gelişmiş veri yapılarını ve iç içe sözlük mimarisini kullanarak çok boyutlu veri kümelerini organize etmek, .get() metoduyla güvenli veri erişimi sağlamak ve sözlük içeriğine dinamik veri ekleme işlemlerini yapılandırmak

Problem Senaryosu: Kod-Asistan projesinin yeni sürümünde, kullanıcıların kütüphanelerindeki edebi eserleri düzenli bir hiyerarşiyle saklayacak bir Dijital Eser Kataloğu veri yapısına ihtiyaç duyulmaktadır. Sistemin kararlı çalışabilmesi için eser isimlerinin ana anahtar olarak kullanıldığı, yazar ve tür gibi özelliklerin ise alt sözlüklerde gruplandığı çok boyutlu bir yapının rastgele veri girişiyle değil önceden tasarlanmış bir plana bağlı kalınarak kurgulanması gerekmektedir. Ayrıca kullanıcıların sistemde henüz tanımlanmamış bir özelliği aradığında programın hata vermemesi, asistanın çözmesi gereken bir çalışma zamanı hatası olarak öne çıkmaktadır.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Kod Dosyasının Hazırlanması ve İç İçe Sözlük Tasarımı

a) Kodlamaya doğrudan geçmeden önce ana sözlükte kullanılacak üç farklı eseri ve bu eserlere ait alt özellikleri hiyerarşik bir plan dâhilinde zihinsel olarak yapılandırınız.

b) Kod editörünüzde dijital_katalog.py adında yeni bir dosya oluşturunuz.

c) Önceden planladığınız bu mimariye sadık kalarak eser_katalogu adında geniş kapsamlı bir ana sözlük başlatınız.

ç) Her bir eserin yazar bilgisini tutmak için ilgili eser adının değer kısmına yeni birer alt sözlük tanımlayınız.

2. Aşama: Verilere Erişim ve Dinamik Güncelleme

a) Zincirleme anahtar çağrısı (iki köşeli parantez yan yana, örneğin ["Mesnevî"]["yazar"]) kullanarak Mesnevî eserine ait yazar bilgisini terminal ekranına yazdırınız.

b) Ana sözlük içerisindeki birinci esere (Dîvân-ı Hikmet) ulaşıp alt sözlüğüne çalışma zamanında dinamik olarak "tür" anahtarı ekleyiniz ve bu anahtara "Tasavvuf" değerini atayınız.

3. Aşama: Güvenli Sorgulama (.get) ve Dijital Depoya Yükleme

a) Üçüncü eserin (Risaletü'n Nushiyye) alt sözlüğünde henüz tanımlanmamış olan "basim_yili" bilgisini programı çökertmeden (KeyError almadan) sorgulamak için .get() metodunu kullanınız ve sonucu ekrana yazdırınız.

b) Terminalde hatasız çıktı üreten kodunuzu kaydediniz.

1. UYGULAMA

c) Dijital depo üzerindeki Kod-Asistan deponuza dosyayı yükleyiniz ve sürüm notu olarak "İç içe sözlük mimarisi ile Dijital Eser Kataloğu oluşturuldu ve .get() ile güvenli erişim sağlandı." yazıp mesleki gelişiminizi belgelendiriniz.

```
print("=== DİJİTAL ESER KATALOĞU YÜKLENİYOR ===\n")
# 1. AŞAMA: İç İçe Sözlük (Nested Dictionary) Mimarisinin Kurulması
eser_katalogu = {
   "Dîvân-ı Hikmet": {
      "yazar": "Hoca Ahmed Yesevî"
   },
   "Mesnevî": {
      "yazar": "Mevlânâ"
   },
   "Risaletü'n Nushiyye": {
      "yazar": "Yunus Emre"
   }
}
# 2. AŞAMA: Zincirleme Veri Erişimi ve Dinamik Ekleme
ikinci_eser_yazari = eser_katalogu["Mesnevî"]["yazar"]
print(f"Mesnevî adlı eserin yazarı: {ikinci_eser_yazari}")
print("\n[BİLGİ] 'Dîvân-ı Hikmet' eseri için 'tür' bilgisi ekleniyor...")
eser_katalogu["Dîvân-ı Hikmet"]["tür"] = "Tasavvuf"
print(f"Güncel Dîvân-ı Hikmet Kaydı: {eser_katalogu['Dîvân-ı Hikmet']}")
# 3. AŞAMA: .get() Metodu ile Güvenli Veri Sorgulama (KeyError Önleme)
print("\n[SORGULAMA] Risaletü'n Nushiyye eseri için basım yılı aranıyor...")
basim_bilgisi = eser_katalogu["Risaletü'n Nushiyye"].get("basim_yili", "Basım yılı bilgisi
bulunamadı.")
print(f"Basım Yılı: {basim_bilgisi}")
print("================================================")
```

Değerlendirme: Çalışmanızı, karekodda yer alan Öz Değerlendirme Formu'ndaki ölçütlere göre değerlendiriniz. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Öz Değerlendirme Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

İç içe sözlük mimarisi, büyük bir kütüphanede her eserin benzersiz bir koda sahip olması ve bu kodun altında yazar veya tür gibi alt bilgilerin düzenli bir şekilde saklanması mantığına benzetilebilir. Yazılım dünyasında da birbiriyle ilişkili, çok boyutlu ve karmaşık veri kümelerinin birbirine karışmadan tek bir çatı altında hiyerarşik bir düzende organize edilmesi için iç içe sözlük mimarisinden faydalanılır. Bu yapısal hiyerarşi sayesinde, devasa boyutlardaki bilgi sistemlerinde aranan herhangi bir alt veriye doğrudan ve güvenli bir şekilde erişim sağlanır.

2. UYGULAMA

Uygulamanın Adı: KOD-ASİSTAN PROFİL SÖZLÜĞÜ

Uygulama Süresi: 20 Dakika

Problem Tanımı: Kod-Asistan projesinde kullanıcı tercihlerini (ad, dil ve tema) saklamak için bir sözlük veri yapısı kurgulanacaktır. Bu uygulamada, tema ayarının dinamik olarak güncellenmesi sağlanırken sistemde bulunmayan bir anahtara erişildiğinde programın hata vermesini önleyecek güvenli veri erişim mekanizması oluşturulmalıdır.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulama Adımları

a) Kod editöründe kod_asistanim çalışma klasörünüzün içine profil_sozlugu.py adında temiz bir Python kaynak kod dosyası açınız.

b) kullanici_profili adında bir sözlük oluşturunuz.

c) Bu ana sözlüğün içine "Kullanici_1" (veya kendi adınızı) anahtar olarak ekleyip değer kısmına "ad", "dil" (örneğin "Türkçe") ve "tema" (örneğin "Açık") bilgilerini barındıran yeni bir alt sözlük tanımlayınız.

ç) Çalışma zamanında kullanıcının tema tercihini doğrudan atama operatörüyle "Koyu" olarak güncelleyiniz.

d) Sistemde varlığından emin olmadığınız "bildirim_sesi" ayarını KeyError almadan güvenli bir şekilde sorgulamak için ilgili alt sözlük üzerinde .get() metodunu kullanınız ve sonucu ekrana yazdırınız.

e) Kodunuzu terminalde test edip hatasız çalıştırdıktan sonra dijital deponuza giderek "Kullanıcı tercihlerini tutan iç içe profil sözlüğü kurgulandı ve .get() ile güvenli erişim sağlandı." sürüm notuyla yükleyiniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

5.2. DOSYA İŞLEMLERİ (I/O) VE VERİ KALICILIĞI

Program çalışırken değişkenlerde tutulan veriler genellikle geçici bellekte bulunur. Dosyaya veya başka bir kalıcı depolama ortamına yazılmayan bu veriler, program sona erdiğinde kaybolur. Gerçek dünya projelerinde kullanıcı ayarları, uygulama geçmişi, müşteri bilgileri gibi verilerin kalıcı olması gerekir. Bir defterin sayfalarına not almanın o bilginin yıllarca saklanmasını sağlaması gibi dosya giriş / çıkış [Input/output(I/O)] işlemleri de verilerin kalıcı depolama birimlerinde (Hard Disk veya SSD) saklanmasına olanak tanır.

5.2.1. Dosya İşlemlerinde Temel Mantık ve open() Fonksiyonu

Python'da bir dosyaya veri yazmak veya dosyadan veri okumak için temel yerleşik open() fonksiyonu kullanılır. Dosyalarla çalışırken izlenmesi gereken işlem süreci; dosyayı açmak, gerekli okuma / yazma işlemini gerçekleştirmek ve ardından dosyayı güvenli bir şekilde kapatmak [close()] şeklinde üç aşamadan oluşur.

Örnek Kod Bloku

```
# Bir dosyayı yazma modunda ('w') açma
# Eğer dosya yoksa sıfırdan oluşturulur, varsa eski içeriği tamamen silinir.
dosya = open("kod_asistan_gunluk.txt", "w", encoding="utf-8")
dosya.write("Kod-Asistan hafıza kayıtları başlatıldı.\n")
dosya.close() # İşlem bitince dosya mutlaka kapatılmalıdır.
```

Kod Analizi

- **Yazma Modu ('w')**: Dosyayı "write" (yazma) modunda açar. Bu modun en kritik özelliği, dosya zaten mevcutsa içindeki eski verileri silmesidir (Üzerine yazmasıdır.).
- **Karakter Kodlaması (encoding="utf-8")**: Türkçe karakterlerin (ç, ş, ğ, ö, ı, ü) sabit diskte bozulmadan saklanabilmesi ve okunabilmesi için evrensel UTF-8 standardı tanımlanmıştır.

5.2.2. Dosya Yolu (Path) Yönetimi: Mutlak ve Göreceli Yollar

Bir dosyayı okumak veya diske kaydetmek için işletim sistemine o dosyanın tam olarak nerede bulunduğunun bildirilmesi gerekir. Dosyanın sistem içindeki adresine **dosya yolu (path)** denir. Yazılım projelerinde dosya yolları temelde iki farklı mimariyle yönetilir.

Bunlardan ilki olan mutlak yol (absolute path), dosyanın bilgisayarın kök sürücüsünden (örneğin C:\ veya /Users/) başlayarak bulunduğu tam adresi ifade eder (örneğin C:\Kullanicilar\Belgeler\Kod-Asistan\veriler\profil.json) ancak bu yöntemin kullanılması programın taşınabilirliğini bozar. Kodlar farklı bir bilgisayarda çalıştırıldığında o bilgisayarda birebir aynı klasör yapısı bulunmayacağı için program hata verebilir.

Bu tür hataları engellemek için ikinci mimari olan göreceli yol (relative path) kullanılır. Göreceli yol, genellikle dosyanın geçerli çalışma dizinine göre tarif edilen adresidir. Örneğin çalışan kod dosyasının yanında veriler adında bir alt klasör varsa buraya dosya kaydetmek için uzun bir mutlak yol yazmak yerine sadece veriler/profil.json yazmak yeterli olur. Bu fiziksel adresleme sayesinde proje dosyası haricî bir bellekle veya dijital bir depoyla (GitHub vb.) başka bir bilgisayara taşınmasına olanak tanır.



5.2.3. with open() Bağlam Yöneticisi

Dosyaları manuel olarak open() ve close() ile yönetmek, programın çalışma zamanında bir hatayla çökmesi durumunda dosyanın açık kalmasına neden olabilir. Dosyanın açık kalması, dosya tanıtıcısı gibi sistem kaynaklarının gereksiz yere kullanılmasına veya dosyanın kilitli kalmasına yol açabilir. Kurumsal yazılım mimarilerinde with open() yapısı dosyanın blok sona erdiğinde veya blok içinde bir hata oluştuğunda otomatik olarak kapatılmasını sağlar. Ancak bu yapının bütün dosya hatalarını engellemediği de unutulmamalıdır.

Örnek Kod Bloku

```
# 'with' yapısı ile güvenli ve otomatik dosya yönetimi
with open("kullanici_ayarlari.txt", "w", encoding="utf-8") as ayar_dosyasi:
    ayar_dosyasi.write("Tema: Koyu\n")
    ayar_dosyasi.write("Dil: Türkçe")
```



Bilgi Kutusu

Dijital Dünyanın Karakutuları (Log Kayıtları)

Uçaklardaki karakutuların kaza anında son saniyeye kadar her şeyi kaydettiğini biliyor musunuz? Benzer şekilde devasa yazılım projelerinde, bankacılık uygulamalarında ve siber güvenlik sistemlerinde de Log (İşlem Geçmişi) adı verilen dijital karakutular bulunur. Bir sunucuya kimin, ne zaman giriş yaptığı veya hangi sistem hatalarının oluştuğu saniye saniye bu dosyalara yazılır. Sistem yöneticileri bu kayıtları tutarken eski verilerin silinmesine asla izin veremezler. Bu yüzden yazılımcılar; geçmişin üzerine yazıp onu yok eden ('w' write) yöntemler yerine, ekleme (append - 'a') standartlarını kullanırlar. 'a' modu, dosya açılırken mevcut içeriği silmeden yeni veriyi dosyanın sonuna ekler. Siber suçların aydınlatılmasında en büyük ipuçları işte bu silinmeyen dijital karakutulardan elde edilir!

5.2.4. Dosyadan Veri Okuma Yöntemleri

Dosyaya yazılan bilgileri kalıcı bellekten tekrar geçici belleğe almak için okuma modu ('r') kullanılır. Okuma işlemi gerçekleştirilirken bilgisayarın bellek kapasitesine ve okunacak verinin yapısına göre farklı metotlar tercih edilir.

Dosya içerisindeki bilginin tamamını tek seferde okumak için read() metodu kullanılır. Bu metot, dosyanın içindeki tüm verileri tek bir karakter dizisi olarak belleğe aktarır. Ancak çok büyük boyutlu dosyalarda tüm veriyi tek seferde belleğe yüklemek, sistem kaynaklarını zorlayabilir ve bellek yetersizliğine yol açabilir. Bu tür büyük hacimli dosyalarda geçici belleği aşırı yüklememek amacıyla readline() metodu devreye girer. Bu metot, dosyanın tamamını otomatik olarak satır satır okumaz; her çağrıda bir satır döndürür ve sadece o anki satırı belleğe taşıyarak güvenli bir veri akışı sağlar.

Dosya içerisindeki verilerin bir döngü yardımıyla sırayla işlenmesi veya gruplandırılması isteniyorsa readlines() metodu kullanılabilir. Bu metot, dosyadaki metni okuyarak her bir satırı bir listenin bağımsız elemanı hâline getirir. Ancak readlines() bütün satırları liste hâlinde belleğe yüklediği için büyük dosyalar için readlines() önerisi uygun değildir.

Örnek Kod Bloku

```
# Dosyayı okuma modunda (‘r’) güvenli bir şekilde açma
with open("kullanici_ayarlari.txt", "r", encoding="utf-8") as ayar_dosyasi:
    okunan_veriler = ayar_dosyasi.read()
    print("--- KAYITLI SİSTEM AYARLARI ---")
    print(okunan_veriler)
```

Terminal Çıktısı

--- KAYITLI SİSTEM AYARLARI ---
Tema: Koyu
Dil: Türkçe

Kod Analizi

1. Adım [Güvenli Okuma (‘r’)]: Dosya **read** modunda açılmıştır. Bu modda dosya içeriği değiştirilemez sadece okuma işlemi gerçekleştirilir.

2. Adım (Bütüncül Aktarım): read() metodu sayesinde dosyadaki tüm metin tek seferde kalıcı diskten okunarak geçici bellekteki okunan_veriler değişkenine aktarılmış ve başarıyla terminal ekranına yazdırılmıştır.

3. UYGULAMA

Uygulamanın Adı: KOD-ASİSTAN HAFIZA DOSYASI (LOG) YÖNETİMİ

Uygulamanın Süresi: 30 dakika

Uygulamanın Amacı: Dosya sistemleri ve with open() yapısı kullanılarak verileri sabit diskte kalıcı olarak saklamak, eski verileri silmeden ekleme ('a' modu) yapmak ve dosya okuma esnasındaki hataları (FileNotFoundError) yönetmek

Problem Senaryosu: Kod-Asistan projesinde işlemler geçici bellekte tutulduğu için program kapatıldığında tüm geçmiş silinmektedir. Asistanın geçmişi hatırlayabilmesi için kullanıcı komutlarının asistan_hafiza.txt dosyasına eski kayıtlar bozulmadan kalıcı şekilde kaydedilmesi ve gerektiğinde okunması hedeflenmektedir. Ayrıca dosya henüz oluşturulmamışken okuma yapıldığında, programın hata verip kapanması engellenmelidir. Bir yazılım geliştirici olarak sistem günlüğünü (log) güvenle arşivlemek ve kullanıcı deneyimini bozan çökme risklerini gidermek temel mesleki görevlerden biridir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Hafızayı Okuma ('r' Modu) ve Güvenli İstisna Yönetimi

a) Kod editörünüzde hafiza_yonetimi.py adında yeni bir dosya oluşturunuz.

3. UYGULAMA

b) Programın ilk kısmında geçmiş kayıtları terminal ekranında göstermek için 'asistan_hafiza.txt' dosyasını 'r' modunda açacak kod blokunu tasarlayınız.

c) İşletim sisteminde o an böyle bir dosya yoksa okuma işlemi programın hata vererek aniden sonlanmasına yol açacaktır. Bu zayıflığı önlemek için okuma bloklarınızı try-except bloku içerisine alınız ve dosya yoksa kullanıcıya "Hafıza dosyası henüz oluşturulmamış." uyarısı gösteriniz.

2. Aşama: Geçmişi Kaydetme ('a' Modu)

a) Kullanıcıdan dinamik bir komut veya not almak için input() fonksiyonunu kullanınız.

b) Alınan bu veriyi güvenle arşivleme görevini yerine getirmek üzere with open() kalıbını kullanarak asistan_hafiza.txt dosyasını 'a' modunda açınız ve veriyi ekleyiniz.

3. Aşama: Sürüm Yönetimi (Dijital Depoya Yükleme)

a) Terminalde hatasız çıktı üreten kod dosyanızı kaydedip dijital depo üzerindeki Kod-Asistan deponuza yükleyiniz. Üstlendiğiniz modül geliştirme görevini eksiksiz biçimde sonlandırmak için sürüm notuna "Kalıcı hafıza (log) modülü eklendi ve dosya okuma işlemleri hata yönetimi ile güvence altına alındı." yazarak mesleki sürecinizi tamamlayınız.

```
# ==========================================================================
# KOD-ASİSTAN PROJESİ - KALICI HAFIZA (LOG) VE DOSYA YÖNETİMİ
# Geliştirici: [Öğrenci Adı Soyadı]
# Standart: PEP 8 / I/O İşlemleri ve Exception Handling (Hata Yönetimi)
# ==========================================================================

print("=== KOD-ASİSTAN HAFIZA VE LOG YÖNETİMİ ===\n")

# 1. AŞAMA: Dosyadan Veri Okuma ('r' - Read Modu) ve İstisna Yönetimi
print("\n--- GEÇMİŞ İŞLEM KAYITLARI (HAFIZA DÖKÜMÜ) ---")

try:
    # Dosya okuma modunda açılır. Program çökme riskine karşı 'try' (dene) blokundadır.
    with open("asistan_hafiza.txt", "r", encoding="utf-8") as dosya:
        gecmis_veriler = dosya.read()
        print(gecmis_veriler)

except FileNotFoundError  as e:
    # Eğer asistan_hafiza.txt bilgisayarda bulunamazsa program çökmez, bu bloka düşer.
     print(f"[SİSTEM UYARISI] Hafıza dosyası bulunamadı! Detay: {e}")
```

3. UYGULAMA

```
# 2. AŞAMA: Dosyaya Veri Ekleme (‘a’ - Append Modu)
kullanici_komutu = input("Asistana kaydetmek istediğiniz notu/komutu giriniz: ")

# ‘a’ modu: Dosya yoksa oluşturur, varsa içeriğini silmeden en sona (kuyruğa) ekler.
with open("asistan_hafiza.txt", "a", encoding="utf-8") as dosya:
    dosya.write(f"-> Sistem Kaydı: {kullanici_komutu}\n")

print("\n[BİLGİ] Veriniz sabit diske (asistan_hafiza.txt) kalıcı olarak kaydedildi.")
```

Değerlendirme: Çalışmanızı, karekodda yer alan Öz Değerlendirme Formu'ndaki ölçütlere göre değerlendiriniz. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Öz Değerlendirme Formu’na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

4. UYGULAMA

Uygulamanın Adı: KULLANICI GERİ BİLDİRİM (FEEDBACK) ARŞİVİ

Uygulama Süresi: 20 Dakika

Problem Tanımı: Geliştirilen Kod-Asistan projesinde, kullanıcıların asistan hakkındaki düşüncelerini, şikayetlerini ve önerilerini kalıcı olarak saklayabilecek bir Geri Bildirim Arşivi'ne ihtiyaç duyulmaktadır. Programın, kullanıcıdan dinamik olarak bir geri bildirim metni alması ve bu metni geri_bildirimler.txt adlı fiziksel bir dosyaya eski yorumları silmeden kaydetmesi hedeflenmektedir. Ardından yöneticinin bu yorumları okuyabilmesi için dosyanın satır satır okunarak kayıtların numaralandırılmış biçimde terminale yansıtılması planlanmaktadır. Sistem ilk kez çalıştırıldığında ve henüz hiçbir yorum yapılmamışken yönetici dosyayı okumaya çalıştığında programın hata vererek aniden sonlanması ise dosya yönetiminde çözülmesi gereken bir hata yönetimi problemi olarak öne çıkmaktadır.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulama Adımları

a) Kod editörünüzde kod_asistanim klasörü içerisine geri_bildirim_modulu.py adında yeni bir Python kaynak kod dosyası oluşturunuz.

b) Arşivdeki geri bildirimleri readlines() metoduyla döngü kurarak okuyunuz. Her bir kaydı, bir sayaç yardımıyla numaralandırılmış (örneğin 1. Yorum, 2. Yorum) biçimde terminal ekranında listeleyecek okuma kodunu yazınız.

c) Okuma işlemi sırasında dosya bilgisayarda henüz yoksa program hata vererek kapanacaktır. Sistem kararlılığını sağlama görevinin bir bilinci olarak bu zayıflığı önlemek için okuma işleminin gerçekleştiği with open() blokunu try-except içerisine alınız. FileNotFoundError hatasını yakalayarak ekrana "Sistemde kayıtlı geri bildirim bulunmamaktadır." uyarısını yazdırınız.

ç) Kullanıcıdan asistan hakkındaki geri bildirimini almak için input() fonksiyonunu kullanınız.

d) Arşivleme görevini yerine getirmek üzere with open() bağlam yöneticisini ve 'a' modunu kullanarak alınan bu metni geri_bildirimler.txt dosyasına alt satıra geçme komutuyla (\n) birlikte ekleyiniz. Türkçe karakter kullanmak için encoding="utf-8" parametresini unutmayınız.

e) Programınızı farklı senaryolarla test edip, çalıştığından emin olduktan sonra üstlendiğiniz modül geliştirme görevini eksiksiz biçimde sonlandırmak için dijital deponuza giderek "Geri bildirim arşiv modülü ve dosya işlemleri oluşturuldu." notuyla deponuza yükleyiniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

5.3. VERİ FORMATLARI VE DENETİM

Metin (txt) dosyaları, verileri sadece düz bir karakter dizisi (string) olarak saklar. Karmaşık bir sözlük yapısındaki veriler (örneğin alt sözlükler veya listeler) doğrudan txt dosyasına yazdırılırsa okunurken tekrar programlanabilir nesnelere dönüştürülmesi oldukça zorlaşır. Endüstriyel yazılım projelerinde, API entegrasyonlarında ve veri depolama işlemlerinde verinin anahtar-değer hiyerarşisini kaybetmeden kalıcı hâle getirilmesi için yapılandırılmış veri formatları kullanılır. **JSON (JavaScript Object Notation)**, yapılandırılmış verilerin saklanması ve sistemler arasında aktarılması için yaygın olarak kullanılan metin tabanlı veri formatlarından biridir.

JSON formatı, Python'daki sözlük ve liste yapılarına benzese de önemli farklar barındırır. JSON anahtarları çift tırnaklı metin olmalıdır, tek tırnak geçerli değildir. Ayrıca Python'daki True, False, None yerine true, false, null kullanılır ve tuple gibi bazı Python türlerinin JSON formatında doğrudan karşılığı bulunmaz. Bu nedenle Python projelerinde karmaşık verileri diskte saklamak ve daha sonra belleğe geri almak için JSON tercih edilir.

5.3.1. Sözlük Yapılarını JSON Formatında Saklama ve Okuma

Python'da JSON dosyalarıyla çalışmak için Python standart kütüphanesinde bulunan json modülü projeye dâhil edilir. Verileri dosyaya yazmak için json.dump(), dosyadan okumak için ise json.load() metotları kullanılır. json.load() ile alınan verinin her zaman orijinal Python yapısıyla aynı olacağı düşünülmemelidir, yalnızca JSON tarafından desteklenen veri türleri uygun Python türlerine dönüştürülür. Verilerin JSON formatında saklanmasının avantajları, bir hava durumu simülasyonu sensör verileri üzerinden şu örnekte modellenmiştir:

Örnek Kod Bloku

```
import json
# Hava durumu simülasyonu için anlık sensör verilerini tutan sözlük
hava_durumu_verisi = {
    "sehir": "Ankara",
    "sicaklik_c": 15.5,
    "durum": "Parçalı Bulutlu",
    "sensorler_aktif": True
}
# 1. Aşama: Veriyi JSON formatında dosyaya kaydetme
with open("hava_durumu.json", "w", encoding="utf-8") as dosya:
    json.dump(hava_durumu_verisi, dosya, ensure_ascii=False, indent=4)
    print("[BİLGİ] Hava durumu simülasyon verileri JSON formatında kalıcı olarak diske yazıldı.")
# 2. Aşama: JSON dosyasından veriyi orijinal sözlük yapısıyla belleğe geri alma
with open("hava_durumu.json", "r", encoding="utf-8") as dosya:
    simulasyon_kaydi = json.load(dosya)
    print("\n--- KAYITLI SİMÜLASYON VERİLERİ ---")
    print(f"Şehir: {simulasyon_kaydi['sehir']} | Sıcaklık: {simulasyon_kaydi['sicaklik_c']}°C")
```



Terminal Çıktısı

[BİLGİ] Hava durumu simülasyon verileri JSON formatında kalıcı olarak diske yazıldı.
--- KAYITLI SİMÜLASYON VERİLERİ ---
Şehir: Ankara | Sıcaklık: 15.5°C

Kod Analizi

1. Adım [json.dump() ile Yapılandırılmış Kayıt]: Bu metot, hava_durumu_verisi adlı sözlüğü diske yazarken veri tiplerini (float, bool) korur. ensure_ascii=False parametresi Türkçe karakterlerin (ç, ş, ğ) JSON formatında bozulmasını engellerken indent=4 parametresi verinin 4 boşluklu hiyerarşik bir girintiyle insanlar tarafından kolay okunabilir bir yapıda dosyaya yazılmasını sağlar.

2. Adım [json.load() ile Kayıpsız Geri Yükleme]: Dosyadan okunan veriler, RAM'e rastgele metinler olarak değil doğrudan kullanılabilir bir sözlük objesi olarak geri döner. Bu sayede veriye simulasyon_kaydi['sicaklik_c'] şeklinde anahtarı üzerinden erişilir.

2. ÖRNEK

İÇ İÇE VERİ YAPILARININ JSON FORMATINDA YÖNETİMİ

Kod-Asistan projesinde birden fazla kullanıcıya ait profil ayarlarının (tema, dil, bildirim onayları) program kapatıldığında kaybolmaması için tek bir merkezî dosyada kalıcı olarak saklanması gerekmektedir. Bu karmaşık ve iç içe geçmiş sözlük yapısının, anahtar-değer hiyerarşisi bozulmadan diske yazılması ve okunması test edilecektir.

Örnek Kod Bloku

```
import json
# İç içe sözlük yapısındaki kullanıcı veri tabanı
kullanici_veritabani = {
    "Kullanici_1": {"ad": "Bilge", "tema": "Koyu", "dil": "Türkçe"},
    "Kullanici_2": {"ad": "Kağan", "tema": "Açık", "dil": "İngilizce"}
}
# 1. Aşama: Veriyi hiyerarşik JSON formatında dosyaya kaydetme
with open("kullanici_veritabani.json", "w", encoding="utf-8") as dosya:
    json.dump(kullanici_veritabani, dosya, ensure_ascii=False, indent=4)
    print("[BİLGİ] Kullanıcı veritabanı JSON formatında diske yazıldı.")
# 2. Aşama: JSON dosyasından veriyi orijinal sözlük objesi olarak geri alma
with open("kullanici_veritabani.json", "r", encoding="utf-8") as dosya:
    yuklenen_veriler = json.load(dosya)
    print("\n--- YÜKLENEN VERİ TABANI ---")
    print(f"Kullanıcı 1 Adı: {yuklenen_veriler['Kullanici_1']['ad']}")
    print(f"Kullanıcı 2 Tema: {yuklenen_veriler['Kullanici_2']['tema']}")
```

2. ÖRNEK

Terminal Çıktısı

[BİLGİ] Kullanıcı veritabanı JSON formatında diske yazıldı.
--- YÜKLENEN VERİ TABANI ---
Kullanıcı 1 Adı: Bilge
Kullanıcı 2 Tema: Açık

Kod Analizi

1. Adım [json.dump() ile Yapılandırılmış Kayıt]: Bu metot, hava_durumu_verisi adlı sözlüğü diske yazarken veri tiplerini (float, bool) korur. Kayıt esnasında kullanılan encoding="utf-8" parametresi dosyanın UTF-8 kodlamasıyla yazılmasını, ensure_ascii=False parametresi ise Türkçe karakterlerin (ç, ş, ğ) JSON dosyasında okunabilir biçimde görünmesini sağlar. indent=4 parametresi ise verilerin 4 boşluklu hiyerarşik bir girintiyle insanlar tarafından kolay okunabilir bir formata bürünmesini sağlamıştır.

2. Adım [json.load() ile Veriyi Geri Yükleme]: Dosyadan okunan veriler, RAM'e rastgele metinler olarak değil JSON verisinden yeni bir Python sözlüğü oluşturularak geri döner. JSON tarafından desteklenen sayı, metin, liste, sözlük, Boolean ve null türleri uygun Python türlerine dönüştürülebilir ancak her Python nesnesi kayıpsız saklanamaz. Bu dönüştürme sayesinde iç içe adresleme (["Kullanici_1"]["ad"]) mantığı hatasız şekilde işletilebilmiştir.

5.3.2. Kod Editörü ile Yerel Sürüm Kontrolü (Git) ve GitHub Entegrasyonu

Geliştirilen Kod-Asistan projeleri, internet tarayıcısı üzerinden sürükle-bırak yöntemiyle depoya yüklenebilmektedir ancak Git kullanımının temel avantajı; değişiklik geçmişini, dalları, commitleri ve eşitleme işlemlerini daha sistemli yönetmesidir.

Bilişim dünyasında kodlar, geliştirme ortamının kendi içerisindeki **Kaynak Denetimi (Source Control)** paneli üzerinden yönetilir. Bu sayede kodlar önce bilgisayarda (yerel Git deposunda) kayıt altına alınır, ardından güvenli bir tünel üzerinden doğrudan bulut deposuna eşitlenir (Push edilir.).

Bu profesyonel yaklaşım, önemli belgeleri büyük bir kütüphane arşivine elden gelişigüzel teslim etmek yerine evrakları önce kendi çalışma masanızda (yerel depo) tasnif etmeye ve yalnızca onaylanmış olanları güvenli bir taşıma hattıyla doğrudan ana arşive (bulut deposu) eşitlemeye benzetilebilir. Yazılım dünyasında da sürükle-bırak yönteminin yarattığı denetimsizlikten kurtulmak ve ekip çalışmasını standartlara bağlamak için kaynak kodlar doğrudan kod editörü üzerinden yönetilir. Bu yapısal hiyerarşi sayesinde projeye ait hassas ayarlar veya kişisel veriler küresel depoya sızmadan, henüz bilgisayar aşamasındayken filtrelenerek dijital güvenlik güvence altına alınır.

5. UYGULAMA

Uygulamanın Adı: VS CODE ÜZERİNDEN PROFESYONEL SÜRÜM YÖNETİMİ

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: Git sürüm kontrol motorunu yerel bilgisayara kurmak, VS Code üzerinde yerel depo başlatmak ve projeyi tarayıcıya ihtiyaç duymadan doğrudan editör üzerinden bulut deposuyla eşitlemek

Problem Senaryosu: Kod-Asistan projesi büyürken her yeni özelliğin eklenmesi sırasında tarayıcıyı açıp dosyaları sürüklemek ciddi bir zaman kaybına ve yönetim zorluğuna yol açmaktadır. Geliştirme hızını artırmak ve projenin sürüm yönetimini tamamen VS Code içerisine entegre etmek için sisteme Git motorunun kurulmasına ve editör üzerinden bulut eşitlemesinin (Push) gerçekleştirilmesine ihtiyaç duyulmaktadır.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Git Sürüm Kontrol Sisteminin Kurulumu

VS Code içerisindeki Kaynak Denetimi panelinin aktif çalışabilmesi için bilgisayarınızda Git yazılımının kurulu olması zorunludur.

a) İnternet tarayıcınızdan Git'in resmî dağıtım sitesine (**git-scm.com**) giriniz.

b) İşletim sisteminize uygun olan güncel sürümü bilgisayarınıza indiriniz.

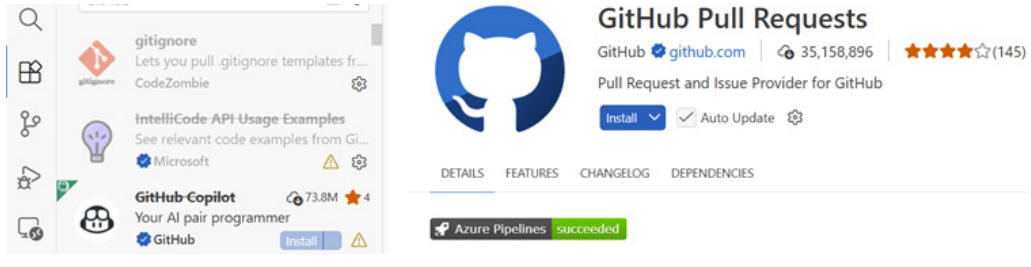
c) İndirilen kurulum dosyasını çalıştırınız. Kurulum ekranlarındaki seçenekleri inceleyerek öğretmeninizin yönlendirmesi doğrultusunda kurulumu tamamlayınız.

2. Aşama: VS Code Git Entegrasyonu ve Eklenti Kurulumu

a) Git kurulumu bittikten sonra açıksa kod editörünüzü tamamen kapatıp yeniden açınız (Bu adım, kod editörünüzün Git motorunu tanıması için kritiktir.).

b) Sol dikey araç panelindeki Eklentiler (Extensions) sekmesine tıklayınız.

c) Arama çubuğuna **GitHub Pull Requests and Issues (Microsoft'un resmî GitHub eklentisi)** yazarak eklentiyi kurunuz (Görsel 5.1). Bu eklenti, VS Code'un GitHub hesabınızla iletişim kurmasını sağlayacaktır.



Görsel 5.1: VS Code Eklentiler sekmesindeki resmî GitHub eklentisi

5. UYGULAMA

3. Aşama: Yerel Depoyu Başlatma (Initialize Repository)

a) VS Code üst menüsünden File > Open Folder diyerek kod_asistanim proje klasörünüzü açtığınızdan emin olunuz.

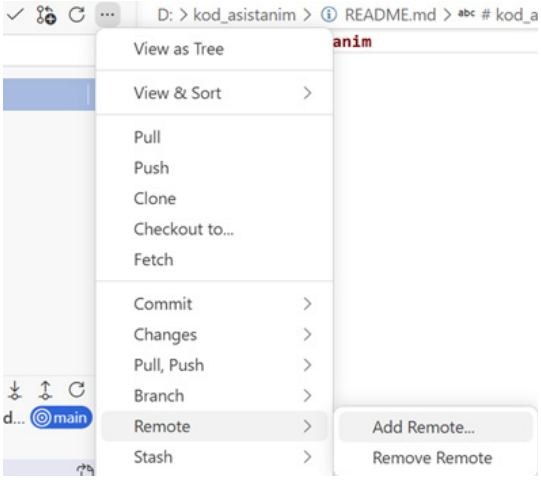
b) Editörün sol dikey menüsünde yer alan, ağ şeklindeki **Kaynak Denetimi (Source Control)** simgesine tıklayınız (Görsel 5.2).



Görsel 5.2: Source Control simgesi

c) Açılan panelde mavi renkli Initialize Repository (Depoyu Başlat) butonuna tıklayınız (Bu işlem, projenizin bulunduğu klasörü yerel bir Git sürüm kontrol merkezine dönüştürür. Klasörünüzün içine gizli bir .git dosyası eklenir.).

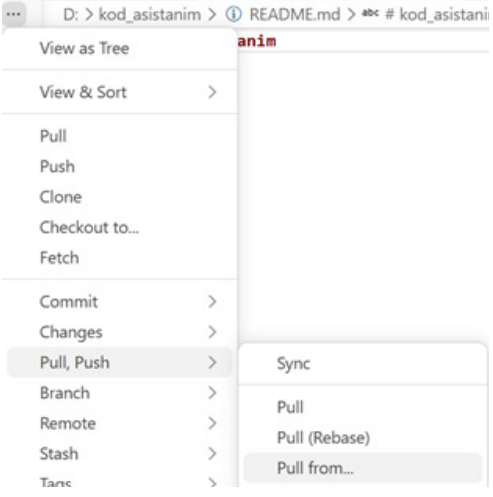
ç) Paneldeki üç nokta ... (Views and More Actions) simgesine tıklayıp, sırasıyla Remote > Add Remote komutunu vererek uzak sunucu ekleyiniz (Görsel 5.3).



Görsel 5.3: Uzak sunucu ekleme

d) Editörün üstünde açılan pencereye GitHub'da oluşturduğunuz depo adresinizi (URL) yapıştırıp Enter'a basınız (örneğin https://github.com/kullanici_adiniz/kod_asistanim). Uzak sunucu adı sorulduğunda varsayılan olarak origin yazıp onaylayınız.

e) Uzak depoda (GitHub'da) önceden oluşturulmuş dosyalar varsa yerel deponuzla eşitlemek için üç noktadan Pull, Push > Pull from... yolunu izleyerek ilgili uzak depoyu seçiniz (Görsel 5.4).



Görsel 5.4: Uzak sunucudan Pull işlemi

5. UYGULAMA

4. Aşama: Değişiklikleri İşleme (Commit) ve Buluta Gönderme (Push / Publish)

a) Depo başlatıldıktan sonra sol panelde dosyalarınızın yanında U [Untracked (İzlenmeyen)] harfi belirecektir. Dosyalarınızı Git takibine (hazırlama alanına) almak için Changes (Değişiklikler) başlığının yanındaki + [Artı (Stage Changes)] simgesine tıklayınız. Dosyaların yanındaki harf A (Added) olarak değişecektir (Görsel 5.5).

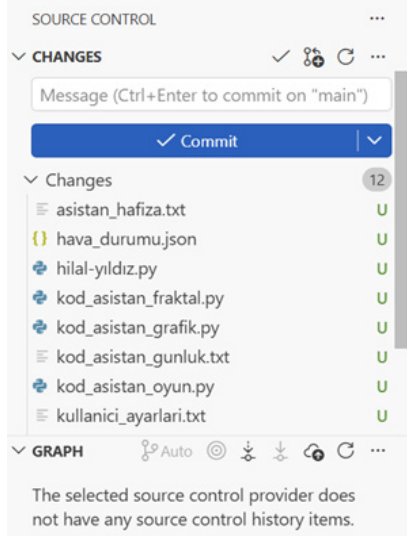
b) Üstteki Message metin kutusuna kurumsal sürüm notunuzu şöyle yazınız: "VS Code Git entegrasyonu sağlandı ve yerel depo başlatıldı."

c) **Commit (İşle)** butonuna basarak değişiklikleri yerel bilgisayarınızda dondurunuz.

ç) Yereldeki kodlarınızı GitHub'a aktarmak için aynı panelde beliren **Publish Branch (Dalı Yayımla)** veya **Sync Changes (Değişiklikleri Senkronize Et)** butonuna tıklayınız.

d) Kod editörü üst kısımda sizden depo hesabınıza giriş yapmanızı (Sign in to GitHub) isteyecektir. Onay vererek tarayıcı üzerinden yetkilendirmeyi tamamlayınız.

e) İşlem tamamlandığında kodlarınız tarayıcıya ihtiyaç duymadan doğrudan VS Code üzerinden GitHub bulut deponuza başarıyla aktarılmış olacaktır.



Görsel 5.5: Source Control paneli

Değerlendirme: Çalışmanızı, karekodda yer alan Öz Değerlendirme Formu'ndaki ölçütlere göre değerlendiriniz. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Öz Değerlendirme Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

5.3.3. Veri Bütünlüğü ve Sürüm Kontrolü: .gitignore Yapılandırması

Yazılım projeleri dijital depolarda açık kaynaklı veya ekip içi kullanıma sunulurken kodların bulunduğu klasördeki her dosyanın buluta yüklenmesi güvenlik zafiyetlerine yol açar. Örneğin kişisel veri, parola, erişim anahtarı veya paylaşılmaması gereken sistem bilgileri içeren dosyalar herkese açık depolara aktarılmamalıdır.

Bir projedeki hangi dosyaların veya klasörlerin sürüm kontrol sistemine dâhil edilmeyeceğini ve dijital depoya aktarılmayacağını belirleyen yapılandırma dosyasına **.gitignore** adı verilir. .gitignore, doğrudan veri bütünlüğü sağlayan veya dosyaları koruyan bir güvenlik sistemi değildir. Git'e hangi izlenmeyen dosya ve klasörleri commit kapsamı dışında tutacağını bildirir.

Uygulama aşamasında, projenin ana dizininde (kodların bulunduğu klasörde) .gitignore (önünde nokta olacak şekilde) adında uzantısız bir metin dosyası oluşturulur. Dijital depoya aktarılması istenmeyen dosyaların isimleri veya uzantıları bu belgeye satır satır yazılır. Burada dikkat edilmesi gereken en önemli sektörel kural, daha önce dijital depoya yüklenmiş olan dosyaların sonradan .gitignore listesine eklense bile geçmiş sürüm kayıtlarından otomatik olarak silinmemesidir. Daha önce Git tarafından izlenmeye başlanmış bir dosya yalnızca .gitignore dosyasına eklenerek izleme kapsamından çıkarılmaz. Önce Git indeksinden kaldırılması gerekir. Bu veri sızıntısı riskini önlemek için yapılandırma işlemlerinin projeye ait kodlar buluta yüklenmeden önce en başından itibaren kurgulanması gerekir.

Örnek .gitignore Dosyası İçeriği

```
asistan_hafiza.txt
geri_bildirimler.txt
hava_durumu.json
profil.json
```

3. ÖRNEK

DİJİTAL VERİ GÜVENLİĞİ VE .gitignore YAPILANDIRMASI

Problem Senaryosu: Kod-Asistan projesi VS Code üzerinden GitHub platformunda eşitlenecektir ancak proje klasörü içerisinde asistanın çalışma geçmişini tutan asistan_hafiza.txt dosyası ve kullanıcıların kişisel profil ayarlarını barındıran veriler adında bir alt klasör bulunmaktadır. Veri güvenliği ilkeleri gereği, bu hassas dosyaların VS Code Kaynak Denetimi paneline düşmesini ve küresel depoya sızmasını engellemek için kurumsal bir güvenlik kalkanı oluşturulması istenmektedir.

Adım Adım Yapılandırma

1. Projenin bulunduğu ana klasörde (çalışma dizininde), isminin başında nokta olan uzantısız bir .gitignore dosyası oluşturulur.
2. İçerisine Git takibinden (sürüm kontrolünden) çıkarılacak dosyaların isimleri veya klasör kuralları yazılıp dosya kaydedilir.

3. ÖRNEK

.gitignore Dosyası İçeriği

```
# 1. Hassas veri ve log dosyalarını gizleme
asistan_hafiza.txt
geri_bildirimler.txt
# 2. 'veriler' adlı klasörü ve içindeki kişisel JSON dosyalarını gizleme
veriler/
*.json
# 3. Geliştirme ortamı (IDE) ve Python önbellek klasörlerini gizleme
.vscode/
__pycache__/
*.pyc
```

Kod Analizi

1. Adım [Görünmezlik (VS Code Tepkisi)]: .gitignore dosyası kaydedildikten sonra, yalnızca daha önce izlenmemiş dosyalar Kaynak Denetimi panelindeki değişiklikler listesinden kaybolur. Editör, bu dosyaları Commit işlemine dâhil etmez. İzlenen dosyalar .gitignore dosyasına eklendikten sonra Kaynak Denetimi panelinden otomatik olarak kaybolmaz.

2. Adım [Klasör Dışlama (/ İşareti)]: Sonuna / (eğik çizgi) konulan isimler (örneğin veriler/), sistem tarafından klasör olarak algılanır. İçinde yüzlerce dosya olsa bile bu klasör bulut yüklemesinden tamamen hariç tutulur.

3. Adım [Joker Karakter (* İşareti)]: Başına yıldız konulan uzantılar (örneğin *.json), dosya adı ne olursa olsun belirtilen izlenmeyen dosyaların yanlışlıkla commit edilmesi riskini azaltır; .gitignore tek başına dosya güvenliği veya gizlilik sağlamaz.

5.3.4. Yapay Zekâ ve Veri Formatı Denetimi

Yapay zekâ asistanları (LLM), karmaşık veri setlerinin JSON formatına dönüştürülmesinde ve .gitignore dosyalarının yapılandırılmasında bir danışman olarak kullanılabilir. Örneğin sisteme entegre edilecek yeni bir veri modelini (XML veya CSV) JSON yapısına çevirmekte zorlanıldığında, yapay zekâya örnek veri seti sunularak kurallara uygun (eksik virgüller veya tırnak işaretlerinden arındırılmış) bir JSON şablonu oluşturması talep edilebilir. İnsan Denetimi prensibi gereği, üretilen şablonun projeye uygunluğu her zaman geliştirici tarafından test edilmelidir.

4. ÖRNEK

YAPAY ZEKÂ DANIŞMANLIĞI İLE CSV'DEN JSON FORMATINA DÖNÜŞTÜRME

Kod-Asistan projesine eski bir yazılımdan alınan kullanıcı iletişim bilgileri aktarılacaktır. Bu veriler, günümüzde de yaygın kullanılan basit, tablo tabanlı bir veri formatı olan CSV (Virgülle Ayrılmış Değerler) formatındadır. Bu karmaşık veri setinin, sistemin anlayabileceği yapılandırılmış JSON formatına dönüştürülmesinde bir Yapay Zekâ (LLM) modeli danışman olarak kullanılacak ve üretilen şablon İnsan Denetimi süzgecinden geçirilecektir. Yapay zekâ aracına gerçek kullanıcı iletişim bilgileri, kişisel veriler veya gizli proje verileri gönderilmemesi gerektiği unutulmamalıdır.

Yapay Zekâya Verilen Komut (Prompt): "Elimde eski sistemden alınmış şu CSV verisi bulunuyor: id, isim, rol 101, Bilge, Yönetici 102, Kağan, Misafir. Lütfen bu veri setini Python'da kullanabileceğim, eksik virgüllerden ve tırnak işaretlerinden arındırılmış kurallara uygun bir JSON şablonu hâline dönüştürür müsün?"

```
{
  "kullanicilar": [
    {
      "id": 101,
      "isim": "Bilge",
      "rol": "Yönetici"
    },
    {
      "id": 102,
      "isim": "Kağan",
      "rol": "Misafir"
    }
  ]
}
```

Yapay Zekâ Çıktısı (JSON Şablonu)

Kod Analizi

1. Adım (Yapay Zekâ Danışmanlığı): Yapay zekâ asistanları (LLM), karmaşık veri setlerinin JSON formatına dönüştürülmesinde son derece hızlı ve verimli bir danışman olarak görev yapmaktadır. CSV formatındaki düz metin, saniyeler içinde Python projelerinde kullanılabilecek hiyerarşik bir sözlük ve liste kombinasyonuna çevrilmiştir.

2. Adım (İnsan Denetimi): İnsan denetimi prensibi gereği yapay zekânın ürettiği bu şablonun projeye uygunluğu her zaman geliştirici tarafından test edilmelidir. JSON denetimi yalnızca bir uzmanın gözle incelemesine bırakılmamalıdır. Söz diziminin gerçekten geçerli olup olmadığı JSON ayrıştırıcısıyla test edilmelidir.

6. UYGULAMA

Uygulamanın Adı: KOD-ASİSTAN JSON TABANLI GÖREV YÖNETİCİSİ

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: Sözlük ve liste gibi JSON tarafından desteklenen veri yapılarını dosyada saklamak ve tekrar Python veri yapılarına dönüştürmek, yapısal özelliklerini koruyarak RAM'e geri almak (json.load) ve üretilen veri formatının doğruluğunu yapay zekâ danışmanlığı ile denetlemek

Problem Senaryosu: Geliştirmekte olduğunuz Kod-Asistan projesine dinamik bir Görev Yöneticisi modülü eklenecektir. Bu modül; görev adı, öncelik durumu ve tamamlanma bilgisini içeren bir sözlük veri yapısı kullanacaktır. Verilerin sadece geçici bellekte kalıp program kapanınca silinmesini engellemek için bu görev sözlüğü gorevler.json dosyasına yazdırılacaktır. Ardından programın başka bir aşamasında bu JSON dosyası okunacak ve son olarak ürettiğiniz JSON yapısının küresel standartlara uygunluğu bir yapay zekâ aracına test ettirilecektir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Sözlük Yapısının Kurulması ve Diske Yazılması

a) Kod editörünüzde gorev_yoneticisi.py adında yeni bir dosya oluşturunuz ve en üst satıra import json modülünü dâhil ediniz.

b) İçerisinde en az iki farklı görev barındıran (örneğin görev adı, öncelik ve durum anahtarları içeren) iç içe geçmiş bir sözlük yapısı kurgulayınız.

c) with open() bağlam yöneticisini kullanarak gorevler.json dosyasını yazma modunda açınız.

ç) json.dump() metodunu kullanarak kurguladığınız sözlüğü diske yazdırınız. Okunabilirliği artırmak ve Türkçe karakterleri korumak için indent=4 ve ensure_ascii=False parametrelerini mutlaka ekleyiniz.

2. Aşama: Verilerin JSON Formatından Geri Yüklenmesi

a) Diske başarıyla yazılan JSON dosyasını bu kez okuma modunda açınız.

b) json.load() metodunu kullanarak dosyadaki verileri RAM'e geri yükleyiniz ve içerisinden belirli bir görevin önceliğini anahtar kullanarak ekrana yazdırınız.

6. UYGULAMA

3. Aşama: Yapay Zekâ ile Veri Formatı Denetimi

a) Bilgisayarınızda oluşan gorevler.json dosyasının içeriğini kopyalayınız.

b) Kullandığınız bir yapay zekâ asistanına şu komutu veriniz: "Aşağıdaki JSON verisini incele. Bu veri; endüstri standartlarına, söz dizimi kurallarına (tırnak işaretleri, virgüller) ve hiyerarşik yapıya uygun mu? Bu veride herhangi bir yapısal hata var mı?". Yapay zekâya gönderilecek görev verilerinde kişisel veya gizli bilgi bulunmaması gerektiği belirtilmelidir.

c) Yapay zekâdan aldığınız doğrulama raporunu (varsa eksiklerinizi) inceleyerek uygulamanızı test ediniz. Ancak JSON dosyasının doğruluğunun yalnızca yapay zekâya test ettirilmesi yeterli değildir. Yapay zekâ görüşü yardımcı değerlendirme olarak kullanılmalı, asıl doğrulama program içinde json.load() metoduyla yapılmalıdır. Hatasız çalışan projenizi dijital deponuza yükleyiniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

JSON formatıyla kaydedilen yapılandırılmış verilerin tek bir ana dizinde yığılması, devasa bir kütüphanedeki tüm kitapların konu raflarına ayrılmadan aynı salona doldurulmasına benzetilebilir. Büyük yazılım projelerinde veri dosyalarının birbirine karışmasını engellemek ve sürdürülebilirliği sağlamak için dosyalar, hiyerarşik alt klasörler içerisinde organize edilir. Bu sistematik yapı, programın farklı bilgisayarlara taşındığında veya bulut üzerinden paylaşıldığında dizin hatası vermeden çalışabilmesi için göreceli dosya yollarıyla kurgulanır.

Kütüphanelerde okuyuculara açık olan genel salonların yanı sıra sadece yetkililerin erişebildiği ve hassas kurumsal belgelerin tutulduğu kapalı arşiv odaları bulunur. Yazılım projeleri genel bulut depolarına aktarılırken veri mahremiyetini sağlamak adına doğrudan bu kapalı arşiv mantığına ihtiyaç duyulur. Kullanıcıların kişisel profil ayarlarını veya sistem günlüklerini barındıran hassas dosyaların herkesin erişimine açık küresel depolara sızması, dijital güvenlik açısından son derece tehlikeli bir zafiyet oluşturur. Bu tür veri sızıntılarını engellemek ve kişisel verilerin gizliliğini korumak amacıyla proje klasörlerinde özel yapılandırma belgeleri kullanılır. Hangi alt klasörlerin veya veri uzantılarının küresel takipten çıkarılacağı kurallara bağlanarak bir koruma kalkanı inşa edilir. Kurulan bu mimari, veriyi sadece başarılı bir şekilde diske yazmayı değil aynı zamanda verinin dijital dünyadaki güvenliğini savunma bilincini yazılım geliştirme sürecinin merkezine yerleştirir.

7. UYGULAMA

Uygulamanın Adı: PROFİL VERİLERİNİN JSON İLE SAKLANMASI VE DOSYA YOLU YÖNETİMİ

Uygulama Süresi: 30 Dakika

Problem Tanımı: Kod-Asistan projenizde kullanıcı profil ayarlarının (isim, tema, dil) kaybolmaması için JSON formatında kalıcı olarak diske yazılması ve okunması istenmektedir. Dosya karmaşasını önlemek adına veriler adında bir alt klasör oluşturarak kayıt işlemini göreceli dosya yolu ile yapmalısınız. Profesyonel bir yazılımcı olarak göreviniz; kişisel verileri barındıran bu JSON dosyasının kazara küresel bulut deposuna sızmasını engellemek için yapay zekâ desteğiyle bir .gitignore yapılandırması gerçekleştirmek ve proje hiyerarşinizi düzenli tutmaktır.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulama Adımları

a) **Kod editörünüzde kod_asistanim ana klasörünüzün içine manuel olarak veriler adında yeni bir alt klasör oluşturunuz.**

b) **Ana dizinde profil_yonetimi.py adında yeni bir Python dosyası oluşturup en üstüne import json modülünü dâhil ediniz.**

c) **İçerisinde "kullanici_adi", "tema" ve "dil" anahtarlarını barındıran bir profil sözlüğü kurgulayınız.**

ç) **with open() bağlam yöneticisini kullanarak oluşturduğunuz sözlüğü JSON formatında veriler klasörünün içine kaydedecek göreceli dosya yolunu ("veriler/profil.json") belirtiniz ve json.dump() metodu ile diske yazdırınız.**

d) **Yazdığınız dosyayı aynı göreceli yol ile okuyarak (json.load()) terminale basan kod blokunu oluşturunuz.**

e) **Yapay Zekâ ile Danışmanlık: Kullandığınız yapay zekâ asistanına "Bir Python projesindeki 'veriler' adlı alt klasörün tamamını ve içindeki tüm .json dosyalarını Git takibinden çıkarmak için .gitignore dosyasına hangi kuralı yazmalıyım?" sorusunu yöneltiniz.**

f) **Yapay zekâdan aldığınız yanıtı İnsan Denetimi süzgecinden geçirerek, ana dizinize bir .gitignore dosyası açıp kuralı uygulayınız.**

g) **Kodlarınızı çalıştırıp profil dosyasının başarıyla okunduğunu teyit ettikten sonra projeyi "JSON profil veritabanı göreceli yolla oluşturuldu ve .gitignore yapılandırması YZ desteğiyle sağlandı." sürüm notuyla dijital deponuza yükleyiniz.**

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Dereceli Puanlama Anahtarı'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

SIRA SİZDE

1. Veri Yönetimi ve Kalıcı Program Yapıları öğrenme birimi boyunca gerçekleştirdiğiniz tüm etkinlikleri, dosya yönetimi süreçlerinizi ve Kod-Asistan çalışmalarınızı gözünüzün önünden geçiriniz.
2. Aşağıda üç farklı geometrik sembolle tanımlanan alanlardaki (Üçgen, Kare, Çember) yönlendirme sorularını dikkatlice okuyunuz.
3. Öğrenme birimi konularıyla ilgili öğrendiklerinizden hareketle kendi öğrenme yolculuğunuzu yansıtan içten ve özgün cevaplarınızı verilen boşluklara yazınız.

ÜÇGEN: Zihnimde Şekillenen 3 Temel Kazanım

Bu öğrenme biriminin sonunda artık tam anlamıyla uzmanlaştığımı düşündüğüm, başkasına hiç zorlanmadan anlatabileceğim **3 temel veri yönetimi ve program kalıcılığı becerisi / bilgisi şunlardır**:

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

KARE: Zihnimi "Köşeye Sıkıştıran" ve Netleşmesi Gereken Pürüzler

Sözlüklerdeki iç içe yapıları kurgularken, dosya işlemlerinde okuma ve yazma / ekleme modlarını yönetirken, try-except ile istisna yönetimi kurarken veya JSON formatıyla verileri diske kaydederken beni hâlâ zorlayan, zihnimde tam olarak oturmayan ve **öğretmenime sormam gereken köşeli / pürüzlü noktalar** şunlardır:

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

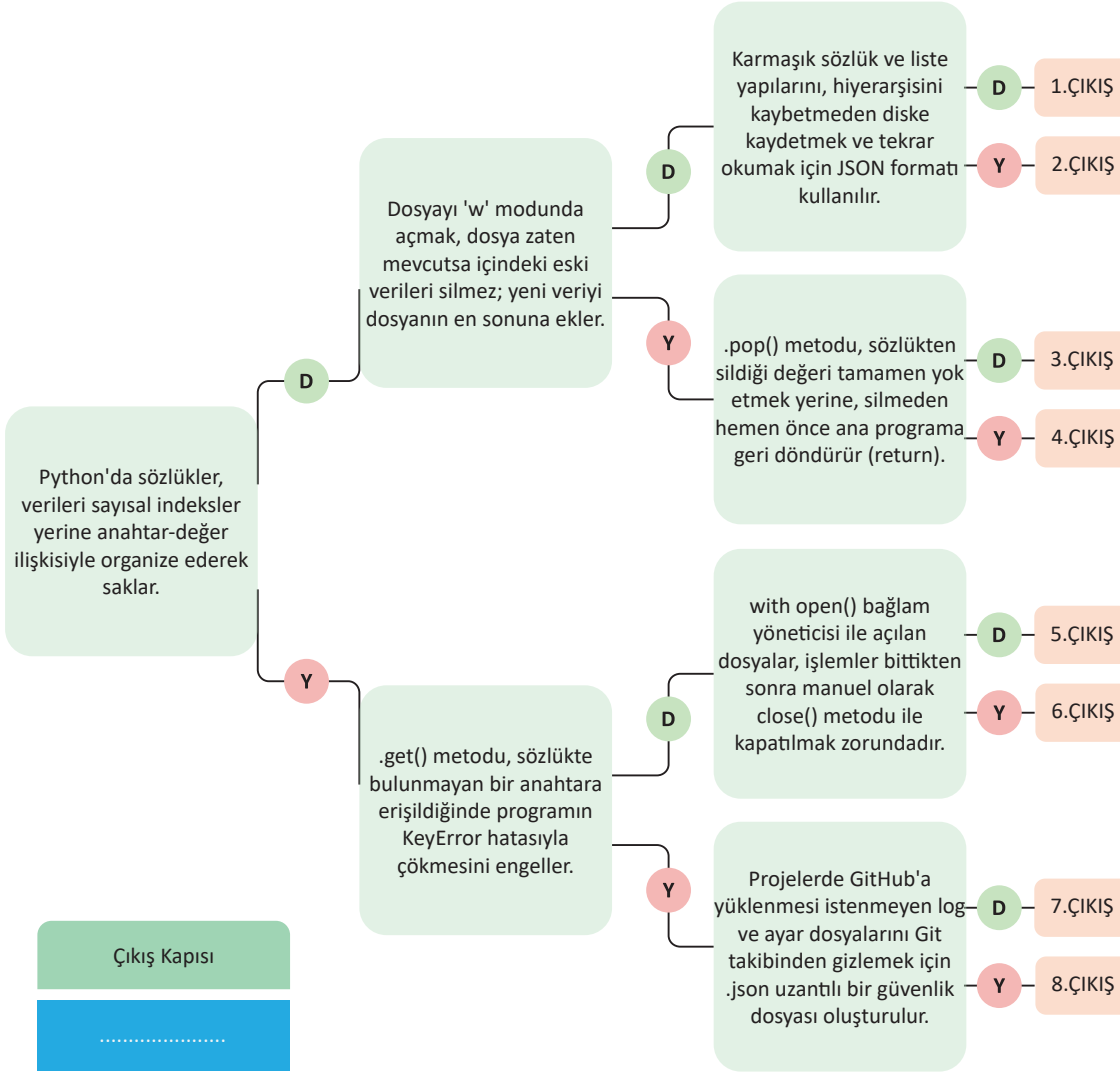
ÇEMBER: Zihnimde Dönen ve Hayatıma Değer Katan Etik Boyut

Kişisel veya hassas verilerin yalnızca gerekli olduğunda saklanması, uygun erişim önlemleriyle korunması ve .gitignore kuralları kullanılarak yanlışlıkla sürüm kontrolüne eklenmesinin önlenmesi hakkında **zihnimde yer eden ve hayatım boyunca dikkat edeceğim temel düşünce** şudur:

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

ÖLÇME VE DEĞERLENDİRME

A) Aşağıdaki Tanılayıcı Dallanmış Ağaç şemasında yer alan 1. ifadeden (kökten) başlayarak okuma yapınız. Okuduğunuz ifadenin doğru (D) veya yanlış (Y) olduğuna karar vererek ilgili ok yönünde (dallanarak) bir sonraki ifadeye ilerleyiniz. Her yol ayrımında verdiğiniz kararlar sizi en son aşamada 8 farklı çıkış kapısından yalnızca birine ulaştıracaktır. Kendi zihinsel mantık akışınızı takip ederek ulaştığınız Çıkış Kapısı Numarasını sayfa altındaki kutucuğa işaretleyiniz.



B) Aşağıda 1'den 9'a kadar numaralandırılmış kutucuklardan oluşan yapılandırılmış grid tablosunu ve altındaki soruları dikkatlice inceleyiniz. Her sorunun cevabı veya cevapları tablodaki kutucuklarda yer almaktadır. Bir kutucuğu birden fazla sorunun cevabı olarak kullanabilirsiniz. Soruların altındaki boşluklara, doğru olduğunu düşündüğünüz kutucuk numaralarını yazınız.

JSON (1)	.get() (2)	'a' Modu (3)
.gitignore (4)	KeyError (5)	with open() (6)
Sözlük (Dictionary) (7)	FileNotFoundError (8)	Göreceli Yol (9)

1. **Yukarıdaki kutucuklardan hangileri, Python programlama dilinde (yanlış adresleme veya yanlış erişimler sonucunda) karşılaşılabilecek çalışma zamanı hata (istisna) türlerindendir?**

   .....

2. **Yukarıdaki kutucuklardan hangileri, Python'da dosya açma, okuma / yazma ve adresleme işlemleri sırasında güvenliği ve taşınabilirliği sağlamak için kullanılan teknik kalıplar, yöntemler veya modlardır?**

   .....

3. **Yukarıdaki kutucuklardan hangileri, anahtar-değer (key-value) ilişkisiyle verileri hiyerarşik olarak gruplandıran gelişmiş veri yapıları veya bu yapılarla doğrudan ilişkili komutlardır?**

   .....

4. **Yukarıdaki kutucuklardan hangileri, bir yazılım projesinde verileri saklamak veya bulut deposundaki (GitHub) gizlilik kurallarını yönetmek için oluşturulan fiziksel dosya türleri veya yapılandırma belgeleridir?**

   .....

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. **Python dilindeki sözlük (dictionary) veri yapısının listelerden (list) ayrılan en temel mimari farkı aşağıdakilerden hangisidir?**
   A) Verileri yalnızca tam sayı türünde saklayabilmesi
   B) İçerisinde sadece tek bir eleman barındırabilmesi
   C) Verilere benzersiz anahtar-değer ilişkisiyle erişilmesi
   D) Çalışma zamanında boyutunun değiştirilememesi
   E) Sadece for döngüleri içerisinde kullanılabilmesi

2. Bir sözlüğün içerisindeki tüm verileri "anahtar" ve "değer" ikilisi olarak gruplandırıp özellikle bir for döngüsünde raporlama yapmak için kullanılan en etkili metot aşağıdakilerden hangisidir?

A) .items()
B) .keys()
C) .values()
D) .pop()
E) .clear()

3.

```
kitap_bilgisi = {"ad": "Mesnevî", "yazar": "Mevlânâ"}
kitap_bilgisi["yazar"] = "Celâleddîn Rûmî"
print(kitap_bilgisi)
```

Verilen Python kodu çalıştırıldığında terminalde oluşan çıktı aşağıdakilerden hangisidir?

A) KeyError hatası verir.
B) {'ad': 'Mesnevî', 'yazar': 'Mevlânâ'}
C) {'ad': 'Mesnevî', 'yazar': 'Celâleddîn Rûmî', 'yazar': 'Mevlânâ'}
D) {'ad': 'Mesnevî', 'yazar': 'Celâleddîn Rûmî'}
E) {'ad': 'Mesnevî'}

4. Yazılım projelerinde oluşturulan asistan_hafiza.txt dosyası okunmak istenmektedir. Eğer bu dosya bilgisayarda henüz yoksa programın çökmesini engellemek için kod blokları hangi koruma mekanizmasının içine alınmalıdır?

A) if-elif-else karar yapısı
B) while True: sonsuz döngüsü
C) try-except FileNotFoundError kalkanı
D) def fonksiyon mimarisi
E) json.dump() metodu

5. with open("rapor.txt", "w", encoding="utf-8") as dosya: satırıyla ilgili aşağıda verilen teknik bilgilerden hangisi yanlıştır?

A) Dosya işlemleri bitince with yapısı sayesinde dosya otomatik olarak kapatılır.
B) 'w' modu eski verileri koruyarak yeni verileri sona ekler ve utf-8 parametresi Türkçe karakterlerin bozulmasını önler.
C) Eğer işletim sisteminde rapor.txt adında bir dosya yoksa sıfırdan yeni bir dosya oluşturulur.
D) Dosyanın içine veri yazmak için dosya.write() metodu kullanılır.
E) Kod blokundaki with open() yapısı, dosya bağlantısını kurumsal standartlarda yöneterek olası program çökmelerini ve bellek sızıntılarını önler.

6. Aşağıdakilerden hangisi json.load() metodunun programlama süreçlerindeki temel işlevidir?

A) Sözlüklerdeki gereksiz verileri hafızadan temizler.
B) Diskteki bir metin dosyasının adını ve dosya yolunu değiştirir.
C) İnternet üzerinden Python yorumlayıcısına yeni kütüphaneler indirir.
D) Dosyayı salt okunur metin (.txt) olarak açıp satır satır terminale yazar.
E) JSON formatında yapılandırılmış bir veriyi bir sözlük objesi olarak yükler.

7. Kurumsal bir yazılım projesi kodlanırken "C:\Kullanicilar\Belgeler\veri.json" gibi mutlak bir dosya yolu yerine doğrudan "veriler/veri.json" gibi göreceli bir dosya yolunun tercih edilmesinin asıl sebebi aşağıdakilerden hangisidir?

A) Diskte daha az depolama alanı kaplaması
B) Dosyanın yapay zekâ tarafından okunmasını engellemesi
C) JSON formatının mutlak dosya yollarını desteklememesi
D) Proje başka bilgisayara taşındığında programın çökmeden çalışması
E) Göreceli yolların sadece Python diline özgü olması

8. Bir geliştirici, sistem ayarlarını barındıran ayarlar.json dosyasını with open() mimarisi ve json.dump() metoduyla diske başarılı bir şekilde kaydetmiştir. Bu projenin bulut deposuna yüklenmesi aşamasında, bu ayar dosyasının dış dünyadan gizlenmesi için dosya adının aşağıdaki yapılandırma dosyalarından hangisinin içine yazılması gerekir?

A) README.md
B) .gitignore
C) main.py
D) requirements.txt
E) config.json

Ç. Aşağıdaki soruların cevaplarını teknik terimleri kullanarak açıklayınız.

1. Aşağıdaki Python kodu çalıştırıldığında sistemin aniden çökerek KeyError fırlattığı gözlemlenmiştir. Bu çalışma zamanı hatasının bellek yönetimi açısından nedenini açıklayınız ve programın çökmemesi için .get() metodunu kullanarak güvenli hâlini yeniden yazınız.

```
kullanici_profili = {"ad": "Bilge", "dil": "Türkçe"}
print("Kullanıcının Seçtiği Tema:", kullanici_profili["tema"])
```

2. **Bir yazılımcı, Kod-Asistan projesinde kullanıcıların her komutunu kalıcı olarak diske kaydetmek amacıyla aşağıdaki kod blokunu kurgulamıştır ancak kod her çalıştığında eski geçmişin tamamen silindiği fark edilmiştir. Bu mantıksal zafiyetin nedenini açıklayarak kod satırındaki hatayı düzeltiniz.**

```
with open("asistan_gecmisi.txt", "w", encoding="utf-8") as dosya:
  dosya.write("Yeni komut işlendi.\n")
```

3. **İşletim sisteminde henüz var olmayan ayarlar.json isimli bir dosyayı doğrudan 'r' modunda açmaya çalışmak programın çökmesine neden olur. Bu dosyanın yokluğu durumunda programın FileNotFoundError fırlatarak kilitlenmesini engellemek için kod blokunu hangi koruma mekanizmasının içine alınmalıdır? Belirlediğiniz koruma mekanizmasını kullanarak güvenli kod mimarisini oluşturunuz.**

4. **Karmaşık ve iç içe geçmiş bir sözlük veri yapısı, düz metin (.txt) yerine neden yapılandırılmış JSON formatında diske kaydedilmelidir? Diske yazdırırken json.dump(), RAM'e geri alırken json.load() metotlarını kullanmanın, veri bütünlüğünü korumasını (veri türlerinin bozulmaması) analitik bir dille tartışınız.**

5. **Kurumsal bir yazılım projesi kodlanırken dosyaların diske kaydedilme aşamasında "C:\Kullanicilar\Belgeler\Proje\veriler\profil.json" gibi mutlak bir dosya yolu yerine "veriler/profil.json" gibi göreceli bir dosya yolu kullanılmasının yazılımın taşınabilirliği üzerindeki etkisini gerekçeleriyle açıklayınız.**

6. **Yazılım projelerinde kişisel ayarları barındıran profil.json veya logları tutan asistan_hafiza.txt gibi hassas dosyaların GitHub vb. küresel dijital depolara kazara yüklenmesini engellemek için hangi güvenlik ve yapılandırma dosyasından faydalanılır? Bu dosyanın dijital veri güvenliği açısından önemini açıklayınız.**



Öğrenme Birimi Karekodu

6. ÖĞRENME BİRİMİ
DAYANIKLILIK, ETİK VE SUNUM

KONULAR

6.1. İSTİSNA YÖNETİMİ VE HATA AYIKLAMA (DEBUGGING)
6.2. BİLİŞİM ETİĞİ VE VERİ GÜVENLİĞİ
6.3. DİJİTAL PORTFOLYO VE PROJE SUNUMU

ANAHTAR KAVRAMLAR

- bilişim etiği
- dayanıklılık
- dijital portfolyo
- hata ayıklama (debugging)
- istisna yönetimi
- KVKK
- proje sunumu
- sürüm (release)
- try-except-finally
- veri güvenliği
- yapay zekâ etiği

BEKLENTİLER

Bu öğrenme biriminde sizden

- istisna yönetimi ve hata ayıklama teknikleriyle dayanıklı kod geliştirmeniz
- yazılım etiği, veri güvenliği ve yapay zekâ güvenliği prensiplerini uygulamanız
- yazılım ürününü teknik ve etik boyutlarıyla sunmanız

beklenmektedir.

BAŞLARKEN

Hiç bankamatikten para çekerken şifrenizi yanlış girdiğinizde makinenin tamamen kapandığını veya çöktüğünü gördünüz mü? Elbette hayır! Sistem sizi uyarır ve işlemi kontrollü bir şekilde yeniden denemenizi ister. Peki, sizin yazdığınız programlar beklenmedik bir hatayla karşılaştığında bu bankamatikler kadar dayanıklı mı?

Dönem başından beri adım adım geliştirdiğiniz Kod-Asistan projenizde artık büyük finale ulaştınız! Bu son aşamada asistanınızı beklenmeyen durumları kontrollü biçimde yönetebilen daha dayanıklı bir yapıya dönüştürmeye, kullanıcı verilerini etik ilkeler ve veri güvenliği kuralları doğrultusunda korumaya ve geliştirdiğiniz bu dijital eseri profesyonel bir portfolyo ile dünyaya sunmaya hazır mısınız?

HAZIRLIK ÇALIŞMALARI

1. Günlük hayatta akıllı telefonunuzda kullandığınız mesajlaşma veya sosyal medya uygulamalarında özellikle hangi kişisel bilgilerinizin korunmasını ve gizli kalmasını beklersiniz?
2. Büyük emek vererek tasarladığınız bir projeyi (örneğin bir yazılımı) sadece kendi bilgisayarınızda saklamak yerine diğer insanlara sunmanın ve onlarla paylaşmanın amaçları ile sağlayacağı faydalar neler olabilir? Açıklayınız.
3. Verilen anahtar kavramın size neleri çağrıştırdığını düşününüz. Bu kavramla ilgili aklınıza gelen en fazla 5 kelimeyi ilgili boşluğa yazınız. Ardından bu kelimeleri kullanarak kavramla ilgili kendi ön bilgilerinizi yansıtan anlamlı bir cümle kurunuz.

Anahtar Kavram: MAHREMİYET

a)

b)

c)

ç)

d)

İlgili Cümlem:

.....

.....

6.1. İSTİSNA YÖNETİMİ VE HATA AYIKLAMA (DEBUGGING)

Yazılım geliştirme sürecinde programların her zaman beklenen koşullarda çalışması mümkün değildir. Kullanıcının hatalı veri girmesi, bir dosyanın bulunamaması veya beklenmeyen bir durumun ortaya çıkması programın normal akışını bozabilir. Bilgisayar bilimlerinde karşılaşılan bu hatalar genel olarak iki grupta incelenir. Kodlama kurallarına uyulmaması (eksik parantez, yanlış yazılmış komut veya hatalı girintileme vb.) sonucu oluşan ve genellikle program çalıştırılmadan önce fark edilen sorunlara **söz dizimi hataları (syntax errors)** denir. Öte yandan, kodun yazımı tamamen doğru olmasına rağmen program çalışırken aniden ortaya çıkan sorunlara ise **çalışma zamanı hataları (runtime errors)** adı verilir. Sayı beklenen yere metin girilmesi veya işletim sisteminde bulunmayan bir dosyanın açılmaya çalışılması çalışma zamanı hatalarına birer örnektir.

Bir yazılımın kalitesi yalnızca doğru sonuç üretmesiyle ölçülemez; gerçek yaşamda kullanıcılar yanlış veri girebilir, dosyalar silinebilir veya sistemler beklenmedik hatalarla karşılaşabilir. Yazılımların işte bu tür kriz durumlarında aniden çökmek yerine kullanıcıya açıklayıcı bilgiler sunarak çalışmaya güvenli bir şekilde devam edebilme özelliğine **dayanıklılık (robustness)** adı verilir. Bu dayanıklılığı sağlamak amacıyla Python programlama dilinde istisna yönetimi (exception handling) mekanizması kullanılır.

OKU-YORUM

Yazılım Dünyasının İlk Gerçek Böceği

Bilgisayar programlarındaki hatalara neden bug (böcek) ve bu hataları tespit edip temizleme işlemine neden debugging (böcek ayıklama) dendiğini hiç merak ettiniz mi? Bu terimlerin bilgisayar dünyasında yaygın olmasının kökeni, bilgisayar bilimleri tarihinin en ilginç olaylarından birine dayanır.

1947 yılında Harvard Üniversitesinde çalışan bilgisayar bilimciler, devasa boyuttaki Mark II bilgisayarının sık sık kilitlendiğini ve hatalı sonuçlar ürettiğini fark ederler. Donanımı detaylıca incelediklerinde bilgisayarın elektromekanik röleleri arasına sıkışmış ve kısa devreye yol açan gerçek bir güve bulurlar. Ekip, bu güveyi cımbızla dikkatlice çıkarıp proje çalışma günlüğüne bantlar ve altına şu tarihi notu düşer: "Bulunan ilk gerçek böcek vakası." O günden beri yazılım dünyasında sistemin beklenmedik şekilde çökmesine, kilitlenmesine veya hatalı çalışmasına yol açan tüm kusurlara **bug**, kodların arasına girerek bu hataları tespit edip temizleme sürecine ise **debugging** adı verilir.

Bugünkü yazılım mühendisleri sistemlerin içinde gerçek böcekler aramasalar da kaynak kodların arasına gizlenmiş söz dizimi ve mantık hatalarını ayıklamak için aynı araştırmacı ruhla hata avcılığı yapmaya devam etmektedir.

Düşünelim-Tartışalım

Yukarıdaki metinden edindiğiniz bilgiler ışığında sıra arkadaşınızla bir araya gelerek aşağıdaki sorular üzerinde tartışınız ve ortak çıkarımlarınızı boşluklara not ediniz.

a) Bir yazılım projesinde kod yazmak ile koddaki hataları arayıp bulma sürecini karşılaştırdığınızda, metinde geçen araştırmacı ruh ve hata avcılığı geliştiriciye hangi problem çözme becerilerini kazandırır? Düşüncelerinizi değerlendiriniz.

Cevap: .....
.....

b) Bilgisayarın kilitlenmesine yol açan yazılımsal bir mantık hatasını bulmak ile donanımdaki fiziksel bir hatayı (örneğin bir böceği) bulma sürecini zorlukları ve farklılıkları açısından karşılaştırınız.

Cevap: .....
.....

6.1.1. Try-Except Yapısı

Hata oluşturma ihtimali bulunan kodlar try bloku içerisine yazılır. Eğer bir hata oluşursa program durmaz, kontrol except blokuna aktarılır ve hata yönetilir. Bazı işlemlerin hata oluşsa da oluşmasa da try blokundan çıkılırken gerçekleştirilmesi gerekir. Bu tür durumlarda finally bloku kullanılır. Bu üçlü koruma mekanizmasına ait temel kullanım şablonu aşağıda gösterilmiştir.

```
# try-except-finally Şablonu
try:
    # Hata üretme riski taşıyan (örneğin tip dönüşümü yapılan veya dosya açılan) şüpheli kodlar
    # (4 boşluk girinti kuralına uyulur.)
    riskli_gorev_satiri_1
    riskli_gorev_satiri_2
except BeklenenHataTuru as e:
    # Belirtilen hata türüyle (Örn: ValueError) karşılaşıldığında program çökmek yerine bu bloka
geçer.
    # Hata nesnesi 'as e' ile yakalanarak Python yorumlayıcısı hata detayı loglanabilir.
    hata_uyarisi_ve_kurtarma_satiri
finally:
    # Hata çıksa da çıkmasa da programın en sonunda mutlaka çalıştırılacak olan temizlik bloku
    # (Opsiyoneldir.)
    temizlik_ve_kapanis_satiri
```

Örnek Kod Bloku

İstisna yönetiminin amacı hataları gizlemek değildir. Amaç, oluşan hataları kontrol altına alarak programın beklenmedik şekilde sonlanmasını önlemektir.

İstisna yönetimi mimarisi, büyük bir binanın elektrik tesisatında aşırı yüklenme meydana geldiğinde, tüm sistemin zarar görmesini engellemek amacıyla sigortaların devreye girerek akımı kontrollü bir şekilde kesmesi mantığına benzetilebilir. Yazılım dünyasında da öngörülemeyen kullanıcı girişleri, dosya arızaları vb. karşısında programın hata vererek aniden sonlanmasını engellemek için try-except koruma kalkanlarından faydalanılır. Bu yapısal kalkan sayesinde kriz anlarında uygulamanın bütünü kilitlenmez ve kullanıcıya sorunun kaynağını açıklayan anlaşılır uyarılar sunularak sistemin kararlılığı korunur.

Geliştirilen bir projede sadece kusursuz veri girişlerinin gerçekleşeceğini varsayarak kod yazmak profesyonel standartlar için yeterli değildir. Herhangi bir verinin hatalı girilmesi veya işletim sistemindeki bir dosyanın silinmesi gibi uç durumların eleştirel bir süzgeçten geçirilerek önceden tahmin edilmesi ve bu risklerin try-except bloklarıyla güvence altına alınması gereklidir.

1. SIRA SİZDE

Bir okul kütüphanesi uygulamasında sistem, kitap ödünç alabilmesi için öğrencilerden sisteme kayıtlı **öğrenci numaralarını** girmelerini talep etmektedir. Bir öğrenci, numara (sayı) yerine klavyeden yanlışlıkla isim bilgisini (harf) girdiğinde, program aniden hata vererek tamamen kapanmaktadır. Sistemin kilitlenmesi nedeniyle kütüphanedeki diğer öğrenciler de uygulamayı kullanamaz hâle gelmiştir.

Yukarıdaki senaryodan hareketle takım arkadaşınızla bir araya gelerek aşağıdaki sorular üzerinde tartışınız. Ortak görüşlerinizi verilen boşluklara açık ve anlaşılır cümlelerle yazınız.

a) Arka planda sayı beklenen bir alana harf girilmesi, sistemde hangi çalışma zamanı hatasını tetiklemiş olabilir? Bu kilitlenme durumunun sadece yanlış veri giren kullanıcının hatası mı yoksa sistemi tasarlayan yazılımcının bir eksikliği mi olduğunu gerekçeleriyle açıklayınız.

Cevap: .....
.....

b) Bir yazılımcı olarak bu kütüphane sistemini tasarlamış olsaydınız programın aniden kapanmasını önlemek için try-except yapısını bu senaryoda nasıl kurgulardınız? Hata yakalandığında kullanıcıya nasıl bir mesaj gösterilmesini sağlardınız?

Cevap: .....
.....

Bir kütüphanedeki otomatik ödünç alma cihazlarına yanlış bir barkod okutulduğunda tüm kütüphanenin hizmet dışı kalması nasıl kabul edilemez bir senaryoysa yazılım dünyasında da hatalı bir veri girişinin tüm programı sonlandırması aynı derecede istenmeyen bir durumdur. Profesyonel bilgi sistemleri, kullanıcıların dalgınlıkla veya bilgisizlikten kaynaklanan hatalı işlemlerini kilitlenmeye terk etmek yerine onları doğru yönlendirecek şekilde tasarlanır.

Bir yazılım geliştirici sadece standart akışlara göre kod yazmakla kalmaz, aynı zamanda olası kriz anlarında neden-sonuç ilişkilerini kurarak hataları güvenle yönetir. Dijital araçları her koşulda ayakta tutmak, bilgi kaybını önlemek ve hizmetlerin kesintisiz çalışmasını sağlamak, üstlenilmesi gereken temel mesleki sorumluluklar arasında yer alır.

1. ÖRNEK

ÇOKLU İSTİSNA YÖNETİMİ VE TRACEBACK ANALİZİ

Bir matematik modülünde kullanıcının girdiği iki sayının birbirine bölünmesi istenmektedir. Kullanıcının sayı yerine harf girmesi (ValueError) veya bir sayıyı sıfıra bölmeye çalışması (ZeroDivisionError) durumlarında programın çökmesini engelleyen güvenli bir yapı kurgulanmıştır.

Örnek Kod Bloku

```
print("--- GÜVENLİ BÖLME İŞLEMİ MODÜLÜ ---")
try:
    # Hata riski taşıyan veri alma ve matematiksel işlem blokları
    bolunen = int(input("Bölünecek sayıyı giriniz: "))
    bolen = int(input("Bölen sayıyı giriniz: "))
    sonuc = bolunen / bolen
    print(f"İşlem Başarılı! Sonuç: {sonuc:.2f}")
# 1. Kalkan: Kullanıcı sayı yerine metin girerse devreye girer.
except ValueError as e:
    print(f"[SİSTEM UYARISI] Lütfen geçerli bir tam sayı giriniz! Detay: {e}")
# 2. Kalkan: Kullanıcı bölen olarak 0 (sıfır) girerse devreye girer.
except ZeroDivisionError as e:
    print(f"[SİSTEM UYARISI] Bir sayı sıfıra bölünemez! Detay: {e}")
# Ne olursa olsun en son çalışacak temizlik bloku
finally:
    print("Bölme işlemi denemesi sonlandı.")
```

Terminal Çıktısı (Kullanıcı Sayı Yerine Harf Girdiğinde)

--- GÜVENLİ BÖLME İŞLEMİ MODÜLÜ ---
Bölünecek sayıyı giriniz: on beş
[SİSTEM UYARISI] Lütfen sadece rakam kullanınız! Detay: invalid literal for int() with base 10: 'on beş'
Bölme işlemi denemesi sonlandı.

Terminal Çıktısı (Kullanıcı Bölen Sayı Olarak 0 Girdiğinde)

--- GÜVENLİ BÖLME İŞLEMİ MODÜLÜ ---
Bölünecek sayıyı giriniz: 12
Bölen sayıyı giriniz: 0
[SİSTEM UYARISI] Bir sayı sıfıra bölünemez! Detay: division by zero
Bölme işlemi denemesi sonlandı.

1. ÖRNEK

Kod Analizi

1. Adım [Çoklu Kalkan (Multiple Exceptions)]: Programda oluşabilecek hatalar tek bir torbaya atılmamış, ValueError ve ZeroDivisionError olarak ayrı ayrı süzgeçlenerek kullanıcıya hatanın kök nedenine uygun nokta atışı uyarılar sunulmuştur.

2. Adım [Detaylı Hata Ayıklama (as e)]: İşletim sisteminin fırlattığı karmaşık Traceback hata detayı, as e kalıbıyla teknik bir değişkene yakalanarak log kaydına eklenmiş ve hata ayıklama standartları sağlanmıştır.

OKU-YORUM

Ariane 5 Uçuş 501 Yazılım Hatası

Yazılım geliştirme süreçlerinde hata yönetimi, yalnızca kodun çalışmasını sağlamakla kalmaz aynı zamanda sistemin bütününe yönelik bir sigorta görevi görür. 1996 yılında fırlatılan Ariane 5 roketinin kalkışından sadece 30 saniye sonra infilak etmesi, bu alandaki ihmallerin büyük maliyetlere ve prestij kayıplarına yol açabileceğine dair önemli bir tarihsel derstir.

Kazanın temelinde, roketin navigasyon sisteminde kullanılan 64 bitlik bir kayan noktalı sayının (floating-point), 16 bitlik tam sayıya (integer) dönüştürülmeye çalışılması yatmaktadır. Bilgisayar sistemlerinde her veri tipinin temsil edebileceği bir sayı aralığı mevcuttur. 16 bitlik bir değişken yalnızca belirli bir büyüklüğe kadar olan değerleri barındırabilir. Roketin ivmelenmesiyle birlikte ortaya çıkan sayısal değer, bu kapasiteyi aştığında sistem bir çalışma zamanı istisnası fırlatmıştır. Sistemin bu kritik modülünde istisna yönetimi mekanizması aktif edilmediği için oluşan hata, sistemin tamamen durmasına ve roketin yönlendirme verilerini kaybetmesine neden olmuştur.

Ariane 5 örneği; bir yazılımın yalnızca doğru çalışmasının yeterli olmadığını, beklenmedik durumlara karşı da dirençli olması gerektiğini kanıtlar. Programlama dillerindeki try-except bloklarının kullanımı, sadece basit bir ValueError yakalamakla sınırlı değildir. Bu yapılar, sistemin kritik bir düğümünde meydana gelen veri kayması, tip dönüşüm hataları gibi öngörülemeyen durumlarda uygulamanın çökmesini engellemek ve güvenli moda geçmesini sağlamak için kurgulanan nihai savunma hattıdır. Eğer Ariane 5'in kod tabanında, bu dönüşüm işlemi bir istisna yönetim bloku içerisinde ele alınmış olsaydı sistem hatayı algılayıp güvenli bir yedekleme moduna geçebilirdi.

OKU-YORUM

Düşünelim-Tartışalım

Verilen metinden edindiğiniz bilgilerden hareketle sıra arkadaşınızla bir araya gelerek aşağıdaki durumlar üzerinde tartışınız ve ortak çıkarımlarınızı verilen boşluklara not ediniz.

a) Ariane 5 kazasını göz önünde bulundurarak bir yazılım projesinde programın doğru çalışması ile programın dayanıklı olması arasındaki yapısal farkı değerlendiriniz.

Cevap: .....

b) Geliştirdiğiniz Kod-Asistan projesinde try-except yapıları kullanmamanın, uygulamanın kararlılığı açısından ne tür riskler oluşturabileceğini gerekçeleriyle açıklayınız.

Cevap: .....

1. UYGULAMA

Uygulamanın Adı: KOD-ASİSTAN DAYANIKLILIK TESTİ VE HATA YÖNETİMİ

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: try-except kontrol bloklarını kullanarak program akışında oluşabilecek çalışma zamanı hatalarını saptamak, hata nesnesini as e ile yakalayarak hata ayıklama yapmak ve sarmal projenin dayanıklılığını artırmak

Problem Senaryosu: Kod-Asistan projesinin v1.0 sürümündeki Vücut Kitle İndeksi (VKİ) Hesaplayıcı modülü başarıyla çalışmaktadır. Test ekibi (QA); kullanıcı boy veya kilo bilgisi girerken, yanlışlıkla harf tuşladığında (örneğin 75 yerine yetmiş beş yazdığında) programın ValueError fırlattığını tespit etmiştir. İstenen bilgiyi 0 (sıfır) girdiğinde ise programın ZeroDivisionError hatası vererek çalışmayı durdurduğu raporlanmıştır. Profesyonel bir yazılımcı olarak göreviniz; "Kullanıcı zaten buraya sayı gireceğini biliyordur." varsayımıyla hareket etmek yerine, kullanıcının dalgınlıkla yanlış bir tuşa basabileceğini öngörerek tüm uç senaryolara karşı sorumluluk almaktır. Bu görev bilinciyle, asistanınızın kullanıcı hataları karşısında aniden kapanmasını engellemek için programın etrafına try-except yapıları kurmalı ve projenizi dayanıklılık testinden geçirmelisiniz.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı öğretmeninizin rehberliğinde gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Hata Ayıklama ve Test Süreci

a) Kod editörünüzde temel Vücut Kitle İndeksi hesaplama kodlarınızı çalıştırınız. Program

1. UYGULAMA

sizden kilo bilgisi istediğinde bilerek sayı yerine metin (örneğin "yetmiş") giriniz ve klavyeden Enter tuşuna basınız.

b) Terminalde beliren hata mesajını inceleyerek sistemin aniden neden çöktüğünü ve karşılaştığınız bu çalışma zamanı hatasının türünü mantıksal olarak analiz ediniz.

c) Programı tekrar çalıştırıp bu kez kilonuzu doğru ancak boy bilginizi 0 (sıfır) olarak giriniz. Ortaya çıkan yeni kilitlenme durumunu değerlendirerek, matematiksel olarak bir sayının sıfıra bölünememesi kuralı ile programın çökmesi arasındaki sebep-sonuç ilişkisini kurunuz.

2. Aşama: Çözüm Üretme ve Güvenli Kod Mimarisi

a) Kod editöründe kod_asistan_dayaniklilik.py adında yeni bir Python kaynak kod dosyası oluşturunuz.

b) Kod editörünüzde temel Vücut Kitle İndeksi hesaplama kodlarını yeni kod dosyası içine kopyalayınız.

c) Yaptığınız bu uygulamalı çıkarımlardan hareketle, kodunuzda hata üretme riski taşıyan (girdi alma ve hesaplama) bölümleri belirleyerek bu riskli satırları try bloku içerisine alınız.

ç) Deneyimlediğiniz her iki çökmeyi de engellemek ve sistemi dayanıklı hâle getirmek için except ValueError as e: ve except ZeroDivisionError as e: bloklarını kurgulayınız. Her iki hata durumunda da kullanıcıya sorunun ne olduğunu anlatan uyarı mesajları yazdırınız.

d) Hesabın başarılı veya başarısız olmasına bakılmaksızın her deneme sonunda çalışması gereken temizlik adımlarını (örneğin "VKİ deneme turu tamamlandı.") bir finally: bloku ekleyerek koda entegre ediniz.

```
# ============================================================
# KOD-ASİSTAN PROJESİ v2.0 - İSTİSNA YÖNETİMİ VE DAYANIKLILIK
# Standart: PEP 8 / try-except-finally ve Hata Ayıklama (Debugging)
# ============================================================
print("=== KOD-ASİSTAN: DAYANIKLI VKİ HESAPLAMA MODÜLÜ ===\n")
while True:
    print("------------------------------------------------")
    # 1. AŞAMA: Hata Riski Taşıyan Kodların try Blokuna Alınması
    try:
        girdi = input("Kilonuzu giriniz (Çıkmak için 'q' tuşlayınız): ")
        # Güvenli çıkış protokolü
        if girdi.lower() == 'q':
            print("Sağlıklı günler dileriz! Sistem kapatılıyor...")
```

1. UYGULAMA

```
        break

    kilo = float(girdi)
    boy = float(input("Boyunuzu metre cinsinden giriniz (Örn: 1.75): "))
    vki = kilo / (boy ** 2)
    print(f"-> Başarılı! Vücut Kitle İndeksiniz: {vki:.2f}")
# 2. AŞAMA: Kullanıcı Sayı Yerine Harf Girerse Çalışacak except Bloku
except ValueError as e:
    print("\n[HATA AĞI] Kritik bir hata yakalandı. Sistem çökmesi engellendi!")
    print(f"Kullanıcı Hatası: Sayısal veri beklenen alana geçersiz karakter girildi.")
    print(f"Log Detayı: {e}\n")
    print("Lütfen verileri doğru formatta yeniden giriniz.")
except ZeroDivisionError as e:
    print("\n[HATA AĞI] Kritik bir hata yakalandı. Sistem çökmesi engellendi!")
    print(f"Kullanıcı Hatası: Bölen alanına 0 girildi.")
    print(f"Log Detayı: {e}\n")
    print("Lütfen verileri doğru formatta yeniden giriniz.")
# 3. AŞAMA: Her Koşulda Çalışacak Temizlik Bloku
finally:
    print("[SİSTEM] VKİ deneme turu tamamlandı.")
```

3. Aşama: Kod Editörü ile Sürüm Yönetimi ve Dijital Depoya Aktarım

a) Kod editörü arayüzünün sol dikey menüsünde bulunan **Kaynak Denetimi (Source Control)** paneline giriniz.

b) Tüm testlerden başarıyla geçen kod_asistan_dayaniklilik.py dosyanızın yanındaki **+ (Stage Changes)** simgesine tıklayarak dosyanızı Git hazırlama alanına alınız.

c) İleti (Message) alanına "VKİ modülüne çoklu istisna yönetimi (ValueError ve ZeroDivisionError) eklendi." sürüm notunu yazarak **Commit (İşle)** butonuna basınız.

ç) Tarayıcıya ihtiyaç duymadan, doğrudan paneldeki **Sync Changes (Değişiklikleri Senkronize Et)** butonuna tıklayarak projenizi dijital deponuza Push ediniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

2. UYGULAMA

Uygulamanın Adı: LİSTE İNDEKSİ VE GİRDİ HATALARININ YÖNETİMİ

Uygulamanın Süresi: 20 Dakika

Problem Tanımı: Geliştirmekte olduğunuz Kod-Asistan projesine Günün Hikmeti adında yeni ve kültürel bir modül ekliyorsunuz. Asistanınızın belleğinde Hoca Ahmed Yesevî, Yunus Emre ve Mevlânâ'ya ait birer tane olmak üzere toplamda üç farklı beyit barındıran ve sözün gücü temasını işleyen bir liste (sozler_listesi) bulunmaktadır. Sistem, kullanıcıdan 0, 1 veya 2 numaralarından birini girmesini isteyecek ve listedeki o indeksli beyiti ekrana yazdıracaktır. Kullanıcı klavyeden 5 gibi listede karşılığı olmayan bir numara girdiğinde program doğrudan IndexError: list index out of range hatası vererek çalışmayı durduracaktır. Sizden beklenen, işlemleri ValueError ve IndexError hatalarına karşı try-except yapısıyla yöneterek sistemi kararlı bir şekilde ayakta tutmanızdır.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulama Adımları

a) Kod editörünüzde kod_asistanim klasörü içine kulturel_miras.py adında yeni bir Python kaynak kod dosyası oluşturunuz.

b) Kodun en üstüne 3 elemanlı bir metin listesi tanımlayınız: sozler_listesi = ["Hikmet sözü kıldırır kâfirleri müselmân, / İhlâs ile dinlese erir dağ ile taşlar. (Hoca Ahmed Yesevî)", "Söz ola kese savaşı, söz ola bitire başı; / Söz ola ağılı aşı, bal ile yağ ede bir söz. (Yunus Emre)", "Sözü süz de söyle, gönlü bulandırmasın. / Sözü diz de söyle, kulağa inci diye takılsın. (Mevlânâ)"]

c) while True: döngüsü başlatınız ve riskli veri alma adımlarını bir try: bloku içerisine alınız. Döngü çıkışı için q harfi kullanılacaktır.

ç) Kullanıcıdan okumak istediği beyitin numarasını alınız ve listedeki karşılığını ekrana yazdırınız.

d) Kullanıcının sayı yerine harf girmesi ihtimaline karşı except ValueError as e: yapısını kurgulayarak uygun uyarıyı yazdırınız.

e) Kullanıcının listede olmayan bir numara (örneğin 5 veya 10) girmesi ihtimaline karşı except IndexError as e: yapısını kurgulayınız. Ekrana "Hata: Girdiğiniz numarada bir beyit bulunmamaktadır!" uyarısıyla birlikte hatanın teknik detayını (e) yazdırınız.

f) Kodunuzu çalıştırınız. Farklı değerler girerek sistemin çökmeden çalıştığını gözlemleyiniz.

g) Tüm testlerden geçen dosyanızı, "Hikmet modülüne ValueError ve IndexError istisna yapıları eklendi." iletisiyle Commit ediniz ve Sync Changes butonuyla projenizi deponuza Push ediniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Yazılım dünyasında belirli bir modüldeki (örneğin sadece okuma modülündeki) hataları ayıklamak, o anki krizi çözmek için yeterlidir. Gelişmiş bir yazılım projesi, onlarca farklı modülün birbiriyle haberleştiği canlı bir organizma gibidir. Bir yazılım geliştirici sadece yeni yazdığı kodları korumakla kalmaz, geçmişte yazdığı tüm kod tabanını da belirli aralıklarla gözden geçirerek gözden kaçan güvenlik veya çökme açıklarını tespit eder.

3. UYGULAMA

Uygulamanın Adı: SARMAL PROJEDE DAYANIKLI KOD GELİŞTİRME

Uygulama Süresi: 30 Dakika

Problem Tanımı: Önceki öğrenme birimlerinde adım adım geliştirdiğiniz Kod-Asistan projenizin temel modülleri (hesaplamalar, menüler vb.) başarıyla çalışmaktadır. Bu modüllerin birçoğu; kullanıcının veri girişi sırasında yapabileceği harf tuşlama, boş bırakma, mantıksal sınır ihlalleri gibi hatalara karşı henüz korunmasızdır. Bir yazılımın kalitesini belirleyen önemli faktörlerden biri dayanıklılığıdır. Sizden beklenen, daha önce yazdığınız kaynak kodları gözden geçirerek hata riski taşıyan bölgeleri istisna yönetimi yapılarıyla donatmanız ve asistanınızı çökmelere karşı daha dayanıklı hâle getirmenizdir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulama Adımları

a) Projenizde kullanıcıdan veri alınan [input() fonksiyonunun kullanıldığı] ve ardından matematiksel dönüşüm [örneğin int() veya float()] yapılan en az iki riskli noktayı belirleyiniz.

b) Hatalı veri girişlerinde programın aniden çökmesini önlemek için ilgili kod bloklarını try-except yapısı içerisine alınız.

c) Bir hata oluştuğunda programın kapanmasını engelleyiniz ve kullanıcıya sorunun kaynağını açıklayan anlaşılır bir hata mesajı yazdırınız.

ç) Programın hata sonrasında da normal çalışmasını (menüye veya döngüye devam etmesini) sağlayacak algoritmik düzenlemeyi yapınız.

d) Yaptığınız bu dayanıklılık güncellemelerini test ediniz ve dijital portfolyonuza (dijital deponuza) "Kod-Asistan dayanıklılık yaması uygulandı ve çökme riskleri giderildi." sürüm notuyla birlikte Push ediniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



6.1.2. Traceback Mesajları ve Hata Ayıklama (Debugging)

Bir bilgisayar programı çalışırken beklenmeyen bir krizle karşılaşıp çöktüğünde hiçbir iz bırakmadan aniden ortadan kaybolmaz. Python yorumlayıcısı, programın neden ve nerede durduğunu yazılımcıya anlatmak için terminal ekranında genelikle kırmızı renkli bir hata raporu gösterir. Bu rapora **Traceback (Geriye Dönük İz Sürme)** adı verilir.

Traceback mesajları aslında bilgisayarın sizden yardım isteme şeklidir. Usta bir yazılımcı, bu kırmızı metinleri bir panik sebebi olarak görmek yerine programı iyileştirmek için bırakılmış ipuçları olarak okur. Bir hata mesajı her zaman **aşağıdan yukarıya doğru** okunur. En alt satır hatanın kimliğini, üst satırlar ise hatanın adresini verir.

Örnek Traceback

```
Traceback (most recent call last): File "hesaplayici.py", line 4, in <module>
```

Hatanın hesaplayici.py adlı dosyanın tam 4. satırında gerçekleştiğini gösterir.

Hatalı Kod Satırı

```
yas = int(input("Yaşınızı giriniz: "))
```

Sistem çökerken Python yorumlayıcısının okumaya çalıştığı o sorunlu kod satırını gösterir.

Örnek Traceback

```
ValueError: invalid literal for int() with base 10: 'on beş'
```

Hatanın kök nedenidir. Burada ValueError (Değer Hatası) yaşandığı ve kullanıcının rakam yerine harflerle 'on beş' yazdığı için dönüşümün yapılamadığı söylenmektedir.

Bir matematik sınavında karmaşık bir denklemin sonucu yanlış bulunduğunda, kâğıttaki tüm işlemlerin silinmesi yerine çözüm adımları sondan başa doğru takip edilerek hatanın nerede yapıldığı bulunmaya çalışılır. Yazılım dünyasında program aniden sonlandığında terminal ekranına yansıyan Traceback mesajları da bu mantıkla çalışır. Bu bildirimler bir başarısızlık tablosu değil aksine bir dedektifin karmaşık bir olay yerindeki ipuçlarını sondan başa doğru izleyerek sorunun asıl kaynağına ulaşmasını sağlayan değerli bir yol haritasıdır.

Bir yazılım geliştirici, ekranda beliren bu hata raporlarını panik unsuru olarak görmek yerine titiz bir dedektif gibi o izleri sürerek sistemin nerede ve neden tıkandığını satır satır tespit eder. Beklenmeyen bir veri girişi nedeniyle uygulamanın çalışmayı durdurmasını engellenmek için saptanan zayıf satırlar güvenli koruma blokları (try-except) içine alınır. Böylece hem programın kararlılığı yeniden tesis edilir hem de dijital hizmetlerin kesintisiz çalışması güvence altına alınır.

4. UYGULAMA

Uygulamanın Adı: İZ SÜRME VE HATA AYIKLAMA (DEBUGGING)

Uygulama Süresi: 15 Dakika

Problem Tanımı: Aşağıda, bir yazılımcının kurguladığı basit bir bölme işlemi kodu ve terminalde karşılaştığı Traceback hata mesajı verilmiştir. Bir Hata Avcısı olarak bu kırmızı izleri takip edip problemi tespit etmeniz ve try-except yapısıyla kodun güvenli hâlini yeniden yazmanız beklenmektedir.

Hatalı Kod

```
bolunen = 50
bolen = int(input("Bölen sayıyı giriniz: "))
sonuc = bolunen / bolen
print("Sonuç:", sonuc)
```

Terminal Çıktısı (Traceback)

```
Traceback (most recent call last):
  File "matematik.py", line 3, in <module>
    sonuc = bolunen / bolen
            ~~~~~~~~^~~~~~~
ZeroDivisionError: division by zero
```

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel şekilde gerçekleştiriniz.

Uygulama Adımları

a) Yukarıdaki Traceback mesajını inceleyerek programın tam olarak hangi satırda ve hangi hata türü (Error Type) nedeniyle çöktüğünü saptayarak kâğıdınıza yazınız.

b) Bu hatanın kullanıcıdan kaynaklanan hangi hatalı veri girişi sebebiyle oluştuğunu açıklayınız.

c) Hata riski taşıyan bu bölme işlemini try bloku içine alınız.

ç) Terminalde tespit ettiğiniz hata türünü except ile yakalayarak programın çökmesini engelleyiniz ve ekrana "Bir sayı sıfıra bölünemez, lütfen başka bir rakam giriniz!" şeklinde kararlı bir uyarı mesajı yazdırınız.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



6.2. BİLİŞİM ETİĞİ VE VERİ GÜVENLİĞİ

Geliştirilen yazılımların kod hatalarından arındırılmış ve istisnalara karşı dayanıklı olması, o yazılımın güvenli ve kaliteli bir ürün olması için tek başına yeterli değildir. Teknoloji çağında yazılımlar sadece makinelerle değil insan hayatıyla, kişisel verilerle ve toplumsal yaşamla doğrudan etkileşim içindedir. Bu nedenle bir yazılım geliştiricisinin, kodlama becerisinin yanı sıra bilişim etiği ve veri güvenliği ilkelerine uygun hareket etmesi mesleki bir sorumluluktur.

6.2.1. Veri Güvenliği ve KVKK

Bir mobil uygulama indirildiğinde veya bir web sitesine üye olunduğunda sistem tarafından genellikle ad, soyadı, e-posta adresi, telefon numarası, hatta bazen anlık konum veya sağlık bilgileri gibi kimliğin belirlenmesini sağlayan veriler talep edilir. Bireyi diğerlerinden ayıran tüm bu özgün detaylara **kişisel veri** adı verilir. Bir uygulamanın ne kadar güvenilir olduğu, bu verileri nasıl işlediğiyle doğrudan ölçülür. Etik bir yazılım, bu hassas bilgileri toplamadan önce neyi, neden ve hangi amaçla kullanacağını açıkça belirtmeli ve kullanıcının tamamen kendi özgür iradesiyle verdiği onay olan açık rızasını almalıdır.

Kişisel verilerin bu şekilde şeffafça korunması; kimlik hırsızlığı veya dolandırıcılık gibi siber riskleri en aza indirirken kullanıcıların dijital dünyaya duyduğu güveni artırır. Bu hayati güvenliği sağlamak amacıyla kişisel verilerin korunmasına yönelik tüm yasal süreçler Kişisel Verilerin Korunması Kanunu (KVKK) kapsamında güvence altına alınmıştır. Küresel yazılım ekosisteminde de benzer hassasiyetler geçerlidir. Örneğin Avrupa Birliği ülkelerinde geliştirilen projeler Genel Veri Koruma Tüzüğü (GDPR) adı verilen benzer sıkı kurallarla denetlenir. Toplanan verilerin güvenli biçimde saklanabilmesi ve kullanılabilmesi için şu temel ilkelere dikkat edilmelidir:

Veri Minimizasyonu: Bir yazılım yalnızca amacını gerçekleştirmek için gerekli olan verileri toplamalıdır. Örneğin basit bir hesap makinesi veya el feneri uygulamasının kullanıcının konum bilgisine ya da fotoğraf galerisine erişim istemesi uygun değildir.

Güçlü Parola Kullanımı: Kullanıcı hesaplarının korunması için büyük-küçük harf, rakam ve özel karakterlerden oluşan güçlü parolalar tercih edilmelidir. Ayrıca kullanıcı parolaları doğrudan okunabilir şekilde saklanmamalı, uygun güvenlik yöntemleriyle korunmalıdır.

Veri Paylaşımı ve Gizlilik: Kullanıcılardan elde edilen kişisel veriler, açık rıza olmadan üçüncü kişilerle paylaşılmamalıdır. Verilerin izinsiz paylaşılması hem etik ilkelere hem de yasal düzenlemelere aykırıdır.

6.2.2. Yapay Zekâ Etiği

Yapay zekâ araçları (örneğin büyük dil modelleri), saniyeler içinde binlerce satır kod üretebilen veya hataları bulabilen eş-geliştiriciler olabilir. Bir yazılımın direksiyonunu tamamen bu araçlara teslim etmek büyük teknik ve ahlaki riskler taşır. Bir yazılımın arka planında hangi verilerin toplandığını veya kodların kimler tarafından yazıldığını kullanıcılar çoğu zaman göremez. Bu görünmezlik, yazılımcının omuzlarına büyük bir doğruluk ve güvenilirlik sorumluluğu yükler. Kullanıcılara verilen sözlere sadık kalmak, şeffaf davranmak ve yapay zekâ asistanlarından etik sınırlar çerçevesinde faydalanmak profesyonel bir yazılımcının en temel özelliğidir.

Yapay zekâ araçları yazılım geliştirme süreçlerinde önemli yardımcılar olabilir ancak bu araçlardan elde edilen çıktılar dikkatli şekilde değerlendirilmelidir. Yapay zekâ araçlarından yararlanırken dikkate alınabilecek temel etik ilkeler şunlardır:

- **Telif Hakkı ve Kaynak Kullanımı**: Yapay zekâ tarafından üretilen kod veya içerikler başka kaynaklardan etkilenmiş olabilir. Bu nedenle kullanılan içeriklerin telif haklarına uygun olup olmadığı kontrol edilmelidir.
- **Ön Yargı (Bias)**: Yapay zekâ sistemleri, eğitildikleri verilerde bulunan ön yargıları yansıtabilir. Üretilen sonuçların ayrımcılık içermediğinden emin olunmalıdır.
- **Doğrulama ve İnsan Denetimi**: Yapay zekâ tarafından üretilen bilgiler ve kodlar doğrudan kullanılmamalıdır. Bu çıktılar doğruluk, güvenlik ve etik açıdan mutlaka insan tarafından gözden geçirilmelidir.
- **Şeffaflık ve Dürüstlük**: Bir projede yapay zekâ araçlarından yararlanıldıysa bu durum teknik dokümantasyonda veya proje açıklamalarında açıkça belirtilmelidir. Bu yaklaşım bilimsel dürüstlük ve mesleki etik açısından önemlidir.

Düşünelim-Tartışalım

Bir öğrenci, bilişim teknolojileri dersi için geliştirilmesi istenen proje ödevinin tamamını bir yapay zekâ aracına kodlatmış ve üretilen çıktıları hiçbir değişikliğe uğratmadan veya kontrol etmeden kendi eseri gibi öğretmene teslim etmiştir. Ayrıca projenin sürüm notlarında veya teknik dokümantasyonunda yapay zekâ desteği aldığına dair hiçbir şeffaf beyanda bulunmamıştır.

Yukarıdaki senaryodan hareketle verilen soruları sıra arkadaşınızla değerlendiriniz ve ortak çıkarımlarınızı boşluklara yazınız.

1. Bu davranışın bilişim etiği, şeffaflık ve dürüstlük ilkeleri bağlamında nasıl değerlendirilmesi gerektiğini açıklayınız. Üretilen kodlarda yapay zekâ asistanına atıf yapılmaması, emeğe saygı ilkesi açısından ne tür ihlallere yol açar?

   Cevabınız: .....

2. Yapay zekâ asistanlarından faydalanma sürecinde bir yazılımcının (öğrencinin) mesleki sorumlulukları nelerdir? Yapay zekâdan alınan çıktıların neden doğrudan kullanılmaması ve mutlaka İnsan Denetimi süzgecinden geçirilmesi gerektiğini gerekçelendiriniz.

   Cevabınız: .....

6.2.3. Kullanıcı Girdilerinin Doğrulanması ve Arındırılması

Kullanıcılardan alınan metinlerin içindeki < ve > gibi özel karakterler, web sayfalarında veya veri tabanlarında her zaman sıradan birer metin olarak okunmaz; bazen sistemi yönlendiren gizli birer komut (script) olarak algılanabilir. Kötü niyetli kişilerin sisteme sızmasını ve izinsiz komutlar çalıştırmasını



engellemek amacıyla alınan verilerin diske kaydedilmeden önce bu tür tehlikeli karakterlerden [örneğin .replace() gibi metotlarla] temizlenmesi işlemine **girdi arındırma (sanitization)** adı verilir. Profesyonel ve güvenilir bir yazılım, kullanıcıdan gelen hiçbir veriye doğrudan güvenmez; veriyi işleme almadan önce mutlaka bu güvenlik ve arındırma süzgecinden geçirir.

5. UYGULAMA

Uygulamanın Adı: KULLANICI VERİLERİNİ KORUMA VE KVKK FARKINDALIĞI

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: Kullanıcıdan alınan kişisel verilerin etik kurallar çerçevesinde işlenmesini sağlamak, kullanıcı izni olmadan kalıcı kayıt yapılmasını engellemek

Problem Senaryosu: Kod-Asistan projenizin profil oluşturma modülü (v3.0); kullanıcıdan ad, yaş, e-posta adresi gibi kişisel bilgiler almaktadır. Kişisel verilerin korunmasına ilişkin ilkeler gereği, bu bilgiler kullanıcının açık izni olmadan kalıcı olarak saklanmamalıdır. Bu etkinlikte kullanıcıya bir bilgilendirme metni sunacak, verilerin kaydedilmesi için izin isteyecek ve kullanıcının tercihine göre kayıt işlemini yöneteceksiniz.

Verilen senaryoya göre uygulamanın aşamalarını sırasıyla takip ederek etkinliği gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: Profil Oluşturma ve İzin Kontrolü

a) Kod editörünüzde kod_asistanim klasörü içine veri_guvenligi.py adında yeni bir Python kaynak kod dosyası oluşturunuz.

b) JSON dosyalarıyla çalışabilmek için gerekli kütüphaneyi içe aktarınız.

c) Kullanıcıdan ad, yaş ve e-posta bilgilerini alınız. Girilen verilerin başında veya sonunda bulunabilecek gereksiz boşlukları temizlemek için .strip() metodunu kullanınız.

ç) Ad ve e-posta bilgilerinde <, > karakterlerini .replace() ile temizleyiniz.

d) Bilgileri profil_verisi adlı bir sözlük içerisinde saklayınız.

e) Kullanıcının cevabını .lower() metodu ile filtreleyerek küçük harfe çeviriniz. Cevap "evet" ise verileri json.dump() ile profil.json dosyasına kaydediniz. Cevap "hayır" ise programı güvenli bir else blokuna yönlendirerek, verileri kaydetmeden siliniz.

5. UYGULAMA

2. Aşama: Kodlama ve Test

a) Programı çalıştırarak "evet" seçeneğini test ediniz. profil.json dosyasının oluştuğunu kontrol ediniz.

b) Programı tekrar çalıştırarak "hayır" seçeneğini test ediniz. Bilgilerin kaydedilmediğini doğrulayınız.

```
import json
print("=== KOD-ASİSTAN PROFİL OLUŞTURMA ===\n")
ad_soyad = input("Adınızı ve soyadınızı giriniz: ").strip().replace("<","").replace(">","")
yas = input("Yaşınızı giriniz: ").strip()
eposta = input("E-posta adresinizi giriniz: ").strip().replace("<","").replace(">","")
profil_verisi = {
   "ad_soyad": ad_soyad,
   "yas": yas,
   "eposta": eposta
}
print("\n[VERİ KULLANIM BİLGİLENDİRMESİ]")
print("Girdiğiniz bilgiler yalnızca profil oluşturma amacıyla kullanılacaktır.")
print("Bilgileriniz üçüncü kişilerle paylaşılmayacaktır.")
izin = input(
   "\nBilgilerinizin kaydedilmesini onaylıyor musunuz? (Evet/Hayır): "
).strip().lower()
if izin == "evet":
   with open("profil.json", "w", encoding="utf-8") as dosya:
      json.dump(
         profil_verisi,
         dosya,
         ensure_ascii=False,
         indent=4
      )
   print("\nBilgileriniz profil.json dosyasına kaydedildi.")
else:
   profil_verisi.clear()
   print("\nBilgileriniz kaydedilmemiştir.")
```

5. UYGULAMA

3. Aşama: Kod Editörü ile Sürüm Yönetimi ve Dijital Depoya Aktarım

a) Kod editörü arayüzünün sol dikey menüsünde bulunan Kaynak Denetimi paneline giriniz.

b) Dosyanızı + (Stage Changes) simgesine tıklayarak Git hazırlama alanına alınız.

c) İleti (Message) alanına "Kullanıcı verilerinin gizliliği için KVKK rıza denetimi ve girdi doğrulama (sanitization) katmanı projeye eklendi." yazarak Commit (İşle) butonuna basınız.

ç) Sync Changes butonuna tıklayarak etik kurallara uyumlu projenizi GitHub dijital deponuza Push ediniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Değerlendirme Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Bilişim sistemlerinde kullanıcı verilerini onay alarak kaydetmek, sürecin sadece ilk adımını oluşturur. Elde edilen bu verilere doğrudan güvenmek ve onları olduğu gibi sistemde işlemek, çeşitli dijital riskleri beraberinde getirir. Güvenli bir yazılım mimarisi kurgulanırken, toplanan verilerin sadece yasal standartlara uygun olarak değil siber tehditlere karşı da yapısal olarak güvence altına alınması gereklidir.

Sisteme dâhil edilen her veriye doğrudan güvenmek, güvenlik sınırlarından geçen yolcuların bagajlarını X-ray cihazından geçirmeden uçağa kabul etmeye benzetilebilir. Bilişim sistemlerinde kullanıcılardan alınan metinlerin içerisindeki < ve > gibi özel karakterler, bazen sistemi yönlendiren gizli birer komut (script) olarak algılanır. Sisteme izinsiz sızılmasını ve istenmeyen komutların çalıştırılmasını engellemek amacıyla alınan verilerin diske kaydedilmeden önce tehlikeli karakterlerden [örneğin .replace() gibi metotlarla] temizlenmesine **girdi arındırma (sanitization)** adı verilir.

Tehlikelerden arındırılarak sisteme kabul edilen özellikle parolalar gibi kritik bilgilerin dijital dosyalarda şeffaf bir metin şeklinde bırakılması, farklı bir güvenlik zafiyeti oluşturur. Profesyonel yazılımlarda bu tür hassas veriler, hash (özetleme) adı verilen tek yönlü matematiksel algoritmalarla geri döndürülemez biçimde şifrelenerek saklanır. Veri gizleme mantığını eğitim ortamında kavramak adına gerçek hash algoritmalarına geçmeden önce parolaları, örnek amacıyla yıldız sembolleriyle (*) maskelemek de bu güvenlik zırhının temelini oluşturan bir simülasyon adımıdır. Veri arındırma süzgeçlerini uygulamak ve şifreleri dönüştürerek diske kaydetmek, dijital yapıların güvenliğini sağlamanın standart bir gerekliliğidir.

6. UYGULAMA

Uygulamanın Adı: ŞİFRE MASKELEME VE GÜVENLİ VERİ KAYDI

Uygulamanın Süresi: 20 Dakika

Problem Tanımı: Geliştirmekte olduğunuz Kod-Asistan projesine yeni bir kullanıcı kayıt modülü eklemektesiniz. Sistem kullanıcıdan bir hesap şifresi alacaktır. Siber güvenlik etiği ve veri gizliliği kuralları gereği kullanıcıların belirlediği şifreler sistemin veri tabanına veya JSON dosyalarına asla düz metin olarak kaydedilmemelidir. Gerçek bilgi sistemlerinde parolalar, hash algoritmalarıyla şifrelenerek saklanır. Bu uygulamada ise veri gizleme mantığının daha kolay anlaşılabilmesi amacıyla örnek olarak maskeleme yöntemi kullanılacaktır. Sizden beklenen, kullanıcının girdiği şifrenin uzunluğunu bularak şifreyi aynı uzunlukta yıldız (*) karakterleriyle maskelemeniz ve sadece bu maskelenmiş güvenli veriyi bir JSON dosyasına kaydetmenizdir.

Verilen problem senaryosundan hareketle aşağıdaki işlem basamaklarını sırasıyla takip ederek uygulamanızı bireysel olarak gerçekleştiriniz.

Uygulama Adımları

a) Kod editörünüzde kod_asistanim klasörü içine sifre_maskeleme.py adında yeni bir Python kaynak kod dosyası oluşturunuz ve import json komutunu ekleyiniz.

b) Kullanıcıdan bir şifre girmesini input() fonksiyonu ile talep ediniz.

c) Girilen şifrenin karakter uzunluğunu len() fonksiyonu ile bularak aynı uzunlukta yıldız (*) karakteri üreten matematiksel işlemi kurgulayınız ve sonucu maskeli_sifre adlı yeni bir değişkene atayınız.

ç) Kullanıcının bilgilerini düzenli tutmak için kullanici_hesabi adında bir sözlük yapısı oluşturunuz ve elde ettiğiniz maskelenmiş şifreyi bu sözlüğe değer olarak ekleyiniz.

d) with open() bağlam yöneticisini kullanarak oluşturduğunuz sözlüğü json.dump() metoduyla guvenli_kayit.json dosyasına kalıcı olarak kaydediniz.

e) Kodunuzu çalıştırınız. Terminalden şifrenizi giriniz ve ardından kod editörü sol panelinden oluşan JSON dosyasını açarak şifrenizin açıkça görünmediğini bizzat test ediniz.

f) Kod editörü arayüzündeki Kaynak Denetimi paneline giriniz. Dosyanızı hazırlama alanına alıp "Kullanıcı şifreleri, maskeleme algoritmasıyla güvenli hâle getirilerek JSON formatında kaydedildi." iletisiyle Commit ediniz ve Sync Changes butonuyla projenizi GitHub deponuza Push ediniz.

Değerlendirme: Çalışmanız karekodda yer alan ölçekle değerlendirilecektir. Çalışmanızı planlarken ölçekteki ölçütleri dikkate alınız.

Dereceli Puanlama Anahtarı Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.



Bilgi Kutusu

Sistemlere giriş yaparken ekranda beliren yıldız (*) karakterleri yalnızca yanınızdaki kişinin ekranı okumasını engellemek (maskelemek) içindir. Peki ya siber korsanlar sistemin arka planına sızarsa? Profesyonel yazılımlar ve dev platformlar, parolalarınızı veri tabanına asla doğrudan okunabilir metinler şeklinde kaydetmez. Bunun yerine parolalar, Hash adı verilen tek yönlü ve çok güçlü matematiksel algoritmalarla şifrelenerek saklanır. Girdiğiniz parola; arka planda "8a2b9c..." gibi anlamsız, karmaşık ve geri döndürülemez bir metne dönüştürülerek kilitlenir. Böylece veri tabanı ele geçirilse bile bilgisayar korsanları gerçek şifrenizin ne olduğunu asla çözemezler!

6.3. DİJİTAL PORTFOLYO VE PROJE SUNUMU

Algoritmik tasarım, hata ayıklama, veri yapılandırması ve etik kodlama süreçlerinin ardından geliştirilen bir yazılımın değerini belirleyen son aşama, o yazılımın ne kadar etkili ve profesyonel sunulduğudur. Bilişim dünyasında iyi çalışan bir yazılım, doğru belgelendirilmediğinde ve etkili bir şekilde sunulmadığında yeterince görünür olmayabilir. Bir yazılım geliştiricinin teknik yetkinliği kadar ürettiği çözümü anlatabilme, paylaşabilme ve savunabilme becerisi de önemlidir.

6.3.1. Dijital Portfolyo ve Teknik Dokümantasyon (README.md)

Geliştirilen projelerin tüm kaynak kodlarını barındıran bulut depoları (örneğin GitHub) yalnızca birer yedekleme alanı değil aynı zamanda tüm dünyaya açılan dijital portfolyolardır. Bir projeyi inceleyen başka bir geliştirici, öğretmen veya gelecekteki bir işveren, çoğu zaman karmaşık kaynak kodlarını okumadan önce projenin vitrini niteliğindeki README.md dosyasıyla karşılaşır. Bu nedenle README dosyası; projenin yalnızca adını değil amacını ve mimarisini de açık, düzenli ve şeffaf bilgilerle sunmalıdır. Bir README dokümantasyonu şu temel bölümleri içermelidir:

Proje Amacı ve Mimarisi: Yazılımın temelde hangi sorunu çözdüğü ve modüllerin birbirleriyle nasıl ilişkilendirildiği net bir dille açıklanmalıdır.

Kurulum Adımları: Programın başka bir bilgisayarda sorunsuz çalıştırılabilmesi için gerekli yazılım, kütüphane ve sistem gereksinimleri belirtilmelidir.

Ekran Görüntüleri ve Şemalar: Programın çalışmasına ait arayüz ekran görüntüleri, algoritmayı anlatan akış şemaları veya örnek terminal çıktıları görsel bir kanıt olarak mutlaka eklenmelidir.

Etik Beyan: Kullanıcı verilerinin (KVKK kapsamında) nasıl korunduğu ve kodlama aşamasında yapay zekâ araçlarından destek alındıysa bunun nasıl (İnsan Denetimi süzgeciyle) kullanıldığı bilimsel bir dürüstlükle ve şeffafça beyan edilmelidir.

6.3.2. Final Sürümünün Yayımlanması

Yazılımın temel işlevlerinin kararlı bir şekilde çalıştığı ve kullanıma hazır hâle geldiği aşama **final sürüm** olarak etiketlenebilir. Dijital depoda bir **Release** oluşturmak, projenin belirli bir sürümünü kayıt altına alarak kullanıcıların erişimine sunmaktır.

7. UYGULAMA

KOD-ASİSTAN v3.0 BÜYÜK FİNAL VE YAZILIM LANSMANI

Uygulamanın Süresi: 40 dakika

Uygulamanın Amacı: Sarmal projenin tüm modüllerini içeren profesyonel bir README dokümantasyonu hazırlamak, projenin final sürümünü (Release v3.0) dijital depoda yayımlamak ve tüm gelişim sürecini kapsayan v3.0 final sürümünü dijital portfolyo (GitHub) üzerinden savunmak

Problem Senaryosu: Kod-Asistan projenizin profil oluşturma modülü (v3.0); kullanıcıdan ad, yaş, e-posta adresi gibi kişisel bilgiler almaktadır. Kişisel verilerin korunmasına ilişkin ilkeler gereği, bu bilgiler kullanıcının açık izni olmadan kalıcı olarak saklanmamalıdır. Bu etkinlikte kullanıcıya bir bilgilendirme metni sunacak, verilerin kaydedilmesi için izin isteyecek ve kullanıcının tercihine göre kayıt işlemini yöneteceksiniz.

Dönem boyunca ilmek ilmek işlediğiniz Kod-Asistan projesi; hata ayıklama yapılarıyla, iç içe geçmiş veri sözlükleriyle ve veri güvenliği kurallarıyla donatılarak daha iyi bir mimariye ulaştı. Bir yazılımcı olarak artık bu değerli projeyi endüstri standartlarında bir dokümantasyonla dijital portfolyonuzda sergilemeniz ve sınıf arkadaşlarınızın karşısında bir "Yazılım Lansmanı" ile savunmanız beklenmektedir. Aşağıdaki işlem basamaklarını takip ederek projenizin final sürümünü (v3.0) yayımlayınız ve sınıf içi lansman sunumunuzu gerçekleştiriniz.

Uygulamanın Aşamaları

1. Aşama: README.md Dokümantasyonunun Hazırlanması

a) Kod editörünüzde Kod-Asistan proje klasörünüzü açınız ve ana dizindeki README.md dosyasını açınız.

b) Belgenizi aşağıdaki Markdown şablonuna göre kendi cümlelerinizle doldurarak projenizin vitrinini tasarlayınız.

7. UYGULAMA

```
# KOD-ASİSTAN (v3.0 Final Sürüm)
**Geliştirici:** [Adınız Soyadınız]
## 1. Projenin Amacı ve Mimarisi
Bu proje, kullanıcılara günlük hesaplamalarda yardımcı olan, aynı zamanda verilerini
JSON formatında kalıcı olarak saklayan kararlı bir dijital asistandır. Projede modüler yapı
kullanılmış olup fonksiyonlar farklı dosyalara bölünmüştür.
## 2. Temel Modüller ve Özellikler
* **Çoklu İstisna Yönetimi:** Kullanıcı hatalarında program çökmez, `try-except` bloklarıyla
kararlı çalışır.
* **JSON Tabanlı Veri Saklama:** Kullanıcı tercihleri ve ayarları JSON dosyalarında saklanır.
* **Veri Gizliliği (KVKK):** Kişisel veriler açık rıza olmadan asla diske kaydedilmez ve
bellekten anında silinir.
## 3. Kurulum ve Çalıştırma
Sistemi bilgisayarınızda çalıştırmak için VS Code terminalinde `python kod_asistan_ana.py`
komutunu çalıştırmanız yeterlidir. Hassas ayar dosyaları `.gitignore` ile korunmaktadır.
## 4. Etik Beyan ve Kaynakça
Bu projenin şifreleme algoritmaları tasarlanırken yapay zekâ asistanlarından yardım alınmış
ancak tüm kodlar İnsan Denetimi süzgecinden geçirilerek doğrulanmıştır.
```

2. Aşama: Final Sürümünün (Release v3.0) Yayımlanması

a) Hazırladığınız README dosyasını kaydediniz ve kod editörü Kaynak Denetimi panelini açınız.

b) Tüm değişiklikleri hazırlama alanına alıp "README dokümantasyonu final sürümü için güncellendi." mesajıyla Commit ediniz ve projenizi Push ediniz.

c) İnternet tarayıcısından deponuza giriniz. Sağ taraftaki Releases bölümünden **Create a new release** butonuna tıklayınız.

ç) Etiket (Tag) alanına v3.0, başlık alanına ise Kod-Asistan Büyük Final Sürümü yazarak **Publish release** butonuna basınız ve projenizi yayımlayınız.

3. Aşama: Tüm Gelişim Sürecinin Dijital Portfolyo (GitHub) Üzerinden Savunulması

Projenizin v3.0 final sürümünü başarıyla yayımladıktan sonra aşağıdaki analitik adımları takip ediniz.

a) Release işlemi bittikten sonra GitHub deponuzdaki Source code (zip) bağlantısına tıklayarak projenizin nihai hâlini bilgisayarınıza indiriniz.

b) Arşivdeki kaynak kodların, v3.0 etiketiyle ilişkilendirilen commit içeriğiyle uyumlu olduğunu kontrol ediniz.

7. UYGULAMA

c) Sınıftan sıra arkadaşınızla eşleşiniz. Yalnızca son yazdığınız kodları değil; v0.1 tasarımından başlayarak v3.0 final sürümüne kadar uzanan tüm gelişim sürecinizi, GitHub deponuzdaki sürüm notlarınızı (commit history) ve güncel README.md dosyanızı kanıt olarak göstererek (dijital portfolyonuz üzerinden) arkadaşınıza savununuz. Kod denetçisi rolündeki arkadaşınızdan gelen soruları şeffaflıkla ve mühendislik öz güveniyle cevaplayınız.

ç) Arkadaşınızın dijital portfolyosunu inceleyerek kod kalitesi (değişken isimlendirmeleri, yorum satırları) ve etik şeffaflık (yapay zekâ beyanı vb.) bağlamında ona teknik geri bildirimler (Code Review) veriniz.

Değerlendirme: Yapacağınız lansman ve dokümantasyon çalışması, karekodda yer alan Dereceli Puanlama Anahtarı'na göre değerlendirilecektir. Çalışmanızı planlarken ölçekte yer alan ölçütleri dikkate alınız. Dijital portfolyonuzu yayımlamadan önce kendi sürecinizi denetlemek için diğer karekodda yer alan Öz Değerlendirme Formu'nu doldurunuz.

Dereceli Puanlama Anahtarı Formu'na ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Değerlendirme formuna ulaşmak için karekodu okutunuz.
Dijital yayınlarda karekodun üstüne tıklayarak içeriğe ulaşabilirsiniz.

Bir mimar, boş bir araziye temel atarak başladığı inşaat sürecini yapının kullanıma açıldığı gün tamamlar ve ardından tüm süreci gözden geçirir. Yazılım geliştirme süreci de ürünün dijital depoda yayımlanmasıyla (lansman) olgunluk evresine ulaşır. Temel veri türlerinden başlanıp karmaşık hata yönetimi ve veri güvenliği yapılarıyla donatılan Kod-Asistan projesi, adım adım geliştirilerek profesyonel bir dijital araca dönüşmüştür. Bu uzun süreçte ortaya konulan nihai ürün kadar o ürünü inşa ederken karşılaşılan hatalar, üretilen mantıksal çözümler ve kazanılan dijital tecrübeler de büyük bir mesleki değer taşır.

Geliştirilen profesyonel bir yazılımın son kullanıcıya sunulmasının ardından yazılım geliştiricinin kendi öğrenme yolculuğunu değerlendirmesi ve zihinsel becerilerini tartması beklenir. Tüm tasarım, kodlama ve hata ayıklama adımlarının eleştirel süzgeçten geçirilerek zihinde bıraktığı kalıcı izlerin analiz edilmesi, mesleki gelişimin ve kendini düzenleme becerisinin en temel gerekliliklerinden biridir. Elde edilen bu dijital birikimi somutlaştırmak ve eksiklikleri tespit etmek amacıyla sıradaki yönlendirme adımları takip edilerek öğrenme sürecinin yapılandırılmış bir öz değerlendirmesi yapılmalıdır.

2. SIRA SİZDE

1. Dayanıklılık, Etik ve Sunum öğrenme birimi boyunca gerçekleştirdiğiniz tüm hata ayıklama etkinliklerini, veri güvenliği (KVKK) süreçlerinizi ve Kod-Asistan projenizin v3.0 Final Sürümü lansmanını gözünüzün önünden geçiriniz.
2. Aşağıda üç farklı geometrik sembolle tanımlanan alanlardaki (Üçgen, Kare, Çember) yönlendirme sorularını dikkatlice okuyunuz.
3. Öğrenme birimi konularıyla ilgili öğrendiklerinizden hareketle kendi öğrenme yolculuğunuzu yansıtan içten ve özgün cevaplarınızı verilen boşluklara yazınız.

ÜÇGEN: Zihnimde Şekillenen 3 Temel Kazanım

Bu öğrenme biriminin sonunda artık tam anlamıyla uzmanlaştığımı düşündüğüm, başkasına hiç zorlanmadan anlatabileceğim 3 temel hata yönetimi, bilişim etiği ve sürüm yayınlama becerisi / bilgisi şunlardır:

.....
.....

KARE: Zihnimi "Köşeye Sıkıştıran" ve Netleşmesi Gereken Pürüzler

Hata ayıklama yapılarını kurgularken, girdi arındırma işlemlerini yönetirken, açık rıza döngüsünde belleği .clear() ile temizlerken veya projeyi v3.0 Release paketi olarak dijital depoda yayımlarken beni hâlâ zorlayan, zihnimde tam olarak oturmayan ve öğretmenime sormam gereken köşeli / pürüzlü noktalar şunlardır:

.....
.....

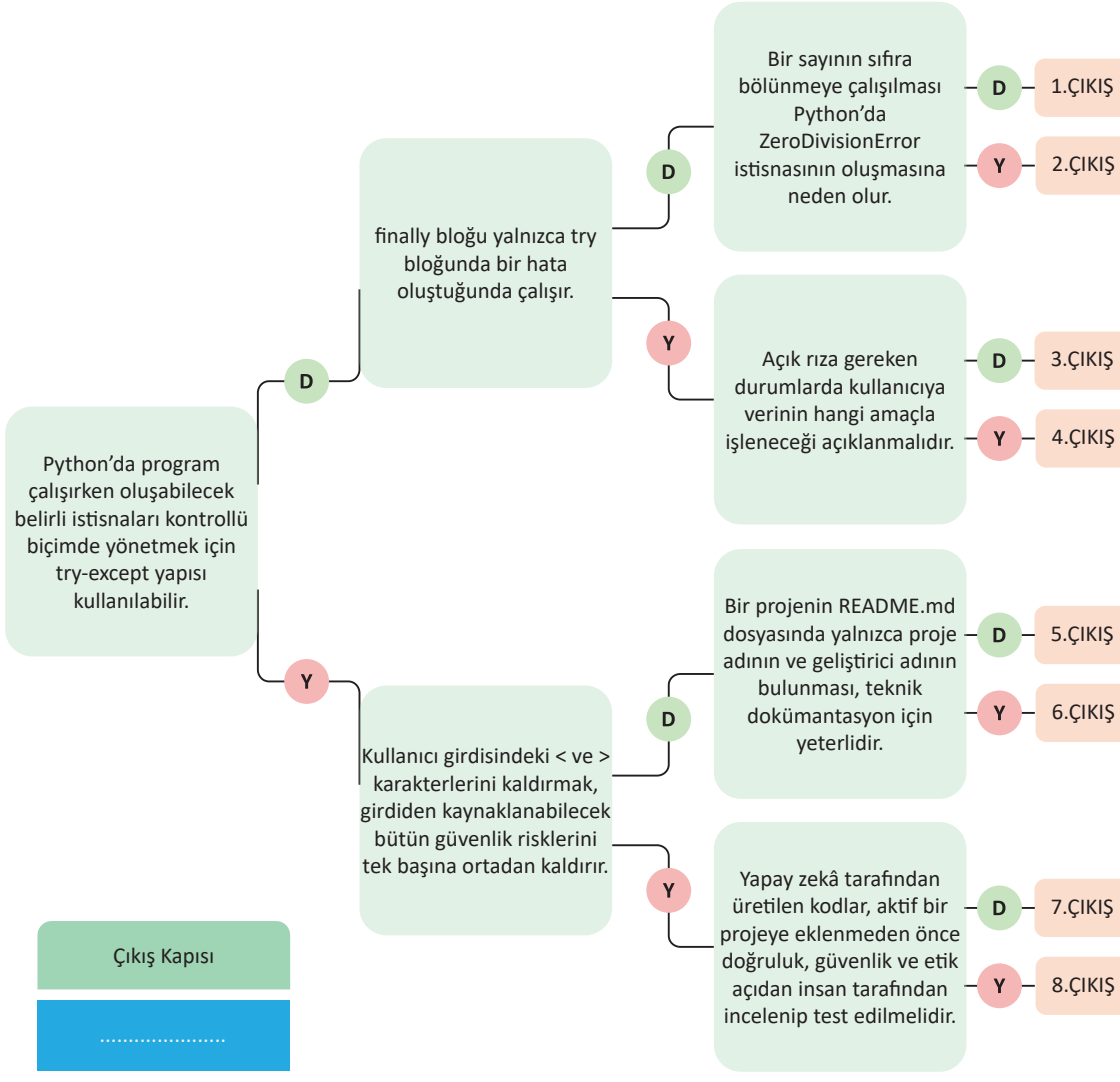
ÇEMBER: Zihnimde Dönen ve Hayatıma Değer Katan Etik Boyut

Yazılım dünyasında kullanıcıların hassas kişisel verilerini şeffaflıkla korumanın önemi, yapay zekâ destekli kodlamada dürüstlük ve kaynak gösterme beyanı ve geliştirdiğim sarmal projeyi tüm sınıfın karşısında savunmanın mesleki öz güvenime kattığı değer hakkında zihnimde yer eden ve hayatım boyunca dikkat edeceğim temel düşünce şudur:

.....
.....

ÖLÇME VE DEĞERLENDİRME

A) Aşağıdaki Tanılayıcı Dallanmış Ağaç şemasında yer alan 1. ifadeden (kökten) başlayarak okuma yapınız. Okuduğunuz ifadenin doğru (D) veya yanlış (Y) olduğuna karar vererek ilgili ok yönünde (dallanarak) bir sonraki ifadeye ilerleyiniz. Her yol ayrımında verdiğiniz kararlar sizi en son aşamada 8 farklı çıkış kapısından yalnızca birine ulaştıracaktır. Kendi zihinsel mantık akışınızı takip ederek ulaştığınız Çıkış Kapısı Numarasını sayfa altındaki kutucuğa işaretleyiniz.



Çıkış Kapısı

.....

ÖLÇME VE DEĞERLENDİRME

B) Aşağıda 1'den 9'a kadar numaralandırılmış kutucuklardan oluşan yapılandırılmış grid tablosunu ve altındaki soruları dikkatlice inceleyiniz. Her sorunun cevabı veya cevapları tablodaki kutucuklarda yer almaktadır. Bir kutucuğu birden fazla sorunun cevabı olarak kullanabilirsiniz. Soruların altındaki boşluklara, doğru olduğunu düşündüğünüz kutucuk numaralarını yazınız.

try-except (1)	IndexError (2)	Sanitization (3)
Release (4)	Finally (5)	README.md (6)
KVKK / GDPR (7)	ValueError (8)	İnsan Denetimi (9)

1. **Yukarıdaki kutucuklardan hangileri Python programlama dilinde karşılaşılabilecek çalışma zamanı hatalarını (istisnalarını) ifade eder?**

   .....
   .....

2. **Yukarıdaki kutucuklardan hangileri, programda oluşabilecek istisnaları yönetmek ve hata oluşup oluşmamasından bağımsız olarak belirli işlemleri gerçekleştirmek için kullanılan kontrol bloklarıdır?**

   .....
   .....

3. **Yukarıdaki kutucuklardan hangileri kişisel verilerin gizliliğini yasal olarak korumak ve kötü niyetli veri girişlerini sisteme sızmadan önce temizlemekle ilgili siber güvenlik / etik kavramlarıdır?**

   .....
   .....

4. **Yukarıdaki kutucuklardan hangileri bir yazılım projesinin dijital portfolyoda teknik olarak belgelendirilmesi ve son hâlinin yayımlanması süreçleriyle ilgilidir?**

   .....
   .....

5. **Yukarıdaki kutucuklardan hangileri, yapay zekâ tarafından üretilen kodların doğruluk ve güvenlik bakımından incelenmesi veya kullanıcı girdilerinin işlenmeden önce arındırılmasıyla ilişkili uygulama ve ilkelerdir?**

   .....
   .....

ÖLÇME VE DEĞERLENDİRME

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Yazılım geliştirme sürecinde çalışma zamanı hatalarının programı çökertmesini engellemek için kullanılan standart hata yönetim mimarisi aşağıdakilerden hangisidir?

A) if-elif-else yapıları
B) while-for döngüleri
C) try-except-finally blokları
D) import ve modül yönetimi
E) json.dump() ve json.load() metotları

2. Üç elemanlı bir liste (örneğin liste = ["A", "B", "C"]) tanımlanmıştır. Kullanıcının liste şeklinde bir eleman çağırmaya çalışması durumunda sınır aşımı nedeniyle oluşan çalışma zamanı hatası aşağıdakilerden hangisidir?

A) ValueError
B) NameError
C) TypeError
D) ZeroDivisionError
E) IndexError

3. try-except hata yönetim mimarisinde finally blokunun temel işlevi aşağıdakilerden hangisidir?

A) Belirli bir hata durumu oluştuğunda devreye girmek
B) Hata oluşmayan durumlarda kod bloklarını çalıştırmak
C) Her koşulda çalışarak bellek temizliğini sağlamak
D) Programı tamamen kapatıp bilgisayarı yeniden başlatmak
E) Oluşan hatayı doğrudan GitHub sunucularına raporlamak

4. Yapay zekâ destekli yazılım geliştirmede karşılaşılabilecek bilişim etiği riskleri arasında aşağıdakilerden hangisi <u>yer almaz</u>?

A) Başkalarına ait lisanslı veya telifli kodların izinsiz kullanılması
B) Modeldeki veri önyargılarının yazılım algoritmalarına doğrudan yansıtılması
C) Üretilen kodların insan denetimi atlanarak aktif sisteme aktarılması
D) Yapay zekâ tarafından üretilen kodların her zaman hatasız çalışması
E) Projede alınan yapay zekâ desteklerinin kullanıcılardan kasten gizlenmesi

5. Sürüm kontrol platformlarında Release oluşturmanın mantığı aşağıdakilerden hangisidir?

A) Projenin silinmiş eski kodlarını kurtarmak
B) Projenin kararlı ve test edilmiş bir sürümünü kullanıcılarla paylaşmak
C) Depoyu diğer yazılımcıların görmesine tamamen kapatmak
D) Kodlarda yeni hatalar oluşturarak denetim sürecini başlatmak
E) Programın çalışma zamanındaki sonsuz döngülerini durdurmak

6. Bir kullanıcının sisteme girdiği "Abc!123" şifresini, JSON dosyasına "*" şekline dönüştürerek kaydetme işlemine bilişim güvenliğinde ne ad verilir?

A) İnsan Denetimi
B) Şifre Maskeleme
C) Tip Dönüşümü
D) Sürüm Kontrolü
E) İstisna Yönetimi

7. Bir yazılımcının kullanıcıdan kişisel verileri almadan önce hukuki bir aydınlatma metni sunması ve verileri kullanıcının onay vermesi durumunda kaydetmesi, hangi yasal / etik kavramla doğrudan ilişkilidir?

A) Release
B) Syntax Checking
C) Açık Rıza ve KVKK Farkındalığı
D) Output Analysis
E) Divide and Conquer

Ç) Aşağıdaki soruları cevaplayınız.

1. Yazılım projelerinde try, except ve finally bloklarının görev hiyerarşisini, programın kararlılığı (çökmeden çalışmaya devam etmesi) bağlamında açıklayınız.

2. Siber güvenlik ve veri bütünlüğü ilkeleri gereği kullanıcının girdiği bir metinsel veriyi doğrudan json.dump() ile diske kaydetmek yerine neden önce Sanitization işleminden geçirilmelidir? Gerekçeleriyle açıklayınız.

3. Bir yazılım mühendisi projesindeki şifreleme ve güvenlik algoritmalarını tamamen bir yapay zekâ asistanına yazdırarak doğrudan aktif sisteme entegre etmiştir. Bu geliştiricinin İnsan Denetimi ve Bilişim Etiği prensiplerine göre yaptığı mesleki ihlalleri açıklayınız.

4. Dijital bir portfolyoda sergilenen projenin vitrini olan README.md dosyasında hangi ana başlıkların bulunması bir zorunluluktur? Projeyi hazırlayan kişinin kariyeri açısından dosyanın önemini açıklayınız.

ÖLÇME VE DEĞERLENDİRME

5. **Programınızda urunler = ["Laptop", "Telefon", "Tablet"] adında bir liste bulunmaktadır. Kullanıcının klavyeden 3 girerek 3. indeksteki ürünü çağırması durumunda sistem neden hata verir? Bu hatanın oluşmasını try-except yapısını kullanarak nasıl engellersiniz?**

6. **Programın dayanıklılığını artırmak ve kullanıcı hatalarından kaynaklanan ani çökmeleri engellemek için kod düzeyinde hangi istisna yönetimi önlemleri alınmalıdır?**

   .....
   .....

7. **Yapay zekâ çıktılarının İnsan Denetimi süzgecinden geçirilmeden doğrudan projede kullanılmasının sistemde oluşturabileceği teknik kusurlar, güvenlik zafiyetleri ve mantık hataları (riskler) nelerdir?**

   .....
   .....

8. **Bir yazılım geliştirme sürecinde etik davranış sergilemenin (açık rıza, veri gizliliği, şeffaflık) ve sistemde alınan yapay zekâ desteklerini proje vitrininde (README) şeffafça belgelendirmenin projenin itibarı açısından önemi nedir?**

   .....
   .....

KAYNAKÇA

Downey, A. B. (2015). Think Python: How to Think Like a Computer Scientist (2nd ed.). O'Reilly Media.

Matthes, E. (2023). Python Crash Course: A Hands-On, Project-Based Introduction to Programming (3rd ed.). No Starch Press.

MEB MTEGM Bilişim Teknolojileri Alanı, Çerçeve Öğretim Programı , Ankara ,2026

Millî Eğitim Bakanlığı. (2026). *Mesleki ve teknik eğitim genel müdürlüğü.*

Severance, A. (2016). Python for Everybody: Exploring Data Using Python 3. CreateSpace Independent Publishing Platform.

Türk Dil Kurumu. (2012). *Türk Dil Kurumu yazım kılavuzu.* Ankara: Türk Dil Kurumu Yayınları.

Türk Dil Kurumu. (2026). *Büyük Türkçe sözlük.*

Türk Dil Kurumu. (2026). *Kısaltmalar.*

Türk Dil Kurumu. (2026). *Noktalama işaretleri (açıklamalar).*

Türk Dil Kurumu. (2026). *Yazım kılavuzu.*

Kaynakça APA 6 kuralları ile düzenlenmiştir.

CEVAP ANAHTARLARI

Cevap anahtarlarına bu karekodu okutarak ulaşabilirsiniz.

Dijital dokümanlarda karekodun üzerine tıklayarak içeriğe ulaşabilirsiniz.



GENEL AĞ, GÖRSEL KAYNAKÇASI

Kitabın görsel kaynakçası, genel ağ kaynakçası ve e-içerik kaynakçasına bu karekodu okutarak ulaşabilirsiniz.

Dijital dokümanlarda karekodun üzerine tıklayarak içeriğe ulaşabilirsiniz.




ROMANYA
UKRAYNA
RUSYA FEDERASYONU
HAZAR DENİZİ
BULGARİSTAN
KARADENİZ
GÜRCİSTAN
YUNANİSTAN
ERMENİSTAN
AZERBAYCAN
İRAN
EGE DENİZİ
MARMARA DENİZİ
AKDENİZ
SURİYE
IRAK
LİBYA
MISIR
İSRAİL
LÜBNAN
FİLİSTİN
Batı Şeria
Doğu Kudüs
Gazze Şeridi
KUZEY KIBRIS TÜRK CUMHURİYETİ
GÜNEY KIBRIS RUM YÖNETİMİ
AĞARA Ö.C.
NAHÇIVAN Ö.C. (AZERBAYCAN)
Bükreş
Sofya
Atina
Beyrut
Şam
Bağdat
Tel Aviv
Lefkoşa
Erenköy
Sinop
Samsun
Trabzon
Rize
Artvin
Ardahan
Kars
Tiflis
Erivan
Iğdır
Ağrı
Erzurum
Erzincan
Giresun
Ordu
Amasya
Tokat
Çorum
Çankırı
Karabük
Kastamonu
Bartın
Zonguldak
Düzce
Bolu
Sakarya
Kocaeli
İstanbul
Tekirdağ
Edirne
Kırklareli
Çanakkale
Gökçeada
Balıkesir
Bursa
Yalova
Bilecik
Eskişehir
Ankara
Kırıkkale
Yozgat
Sivas
Tunceli
Bingöl
Muş
Bitlis
Van
Elazığ
Malatya
Kırşehir
Nevşehir
Kayseri
Aksaray
Konya
Niğde
Kütahya
Afyonkarahisar
Uşak
Manisa
İzmir
Aydın
Denizli
Muğla
Burdur
Isparta
Antalya
Karaman
Mersin
Adana
Osmaniye
Kahramanmaraş
Gaziantep
Şanlıurfa
Kilis
Hatay
Adıyaman
Diyarbakır
Batman
Siirt
Şırnak
Hakkâri
Mardin
Van Gölü
Urumiye Gölü
Tuz Gölü
Marmara Gölü
Yeşilırmak N.
Kızılırmak N.
Sakarya N.
Meriç N.
Büyük Menderes N.
Küçük Menderes N.
Gediz N.
Bakırçay N.
Seyhan N.
Ceyhan N.
Göksu N.
Fırat N.
Dicle N.
Aras N.
Kura N.
Çoruh N.
Süveyş K.
Nil N.
0 50 100 200 300 400 Km
TÜRKİYE'NİN MÜLKİ İDARE BÖLÜMLERİ İLE KARA VE DENİZ KOMŞULARI HARİTASI
BAŞKENT
İL MERKEZİ
GÖL
Ankara
Elazığ
ÜLKE SINIRI
ÖZERK BÖLGE SINIRI
AKARSU


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
Bering Denizi
Ohotsk Denizi
Kuzey Denizi
Akdeniz
Karadeniz
Hazar Denizi
Kızıldeniz
Umman Denizi
Yemen Denizi
Moskova
Astana
Taşkent
Bişkek
Aşkabat
Tahran
Kabil
Yeni Delhi
Ankara
Bakü
Erivan
Tiflis
Bağdat
Şam
Amman
Riyad
Doha
Abu Dabi
Muskat
Sana
Trablus
Tunus
Cezayir
Rabat
Madrid
Paris
Berlin
Varşova
Kiev
Minsk
Helsinki
Oslo
Stockholm
Ulanbatur
Pekin
Delhi
Dakka
Islamabad
Lefkoşa
Beyrut
Kahire
Trablus
Tunus
Cezayir
Rabat
Madrid
Lizbon
Dublin
Londra
Amsterdam
Brüksel
Kopenhag
Viyana
Prag
Budapeşte
Bükreş
Sofya
Belgrad
Saraybosna
Zagreb
Priştine
Tiran
Atina
Lefkoşa
Ankara
İstanbul
İzmir
Cenevre
Roma
Venedik
Milano
Münih
Stuttgart
Frankfurt
Hamburg
Kopenhag
Stockholm
Helsinki
Oslo
Reykjavik
Dublin
Londra
Amsterdam
Brüksel
Paris
Madrid
Lizbon
Rabat
Cezayir
Trablus
Tunus
Kahire
Kızıldeniz
Sudan
Çad
Nijer
Libya
Mısır
Ürdün
Filistin
Suriye
Irak
Kuveyt
Suudi Arabistan
Katar
Birleşik Arap Emirlikleri
Umman
Yemen
Afganistan
Pakistan
Hindistan
Bangladeş
İran
Türkmenistan
Özbekistan
Kırgızistan
Tacikistan
Kazakistan
Rusya Federasyonu
Moğolistan
Çin Halk Cumhuriyeti
Gürcistan
Azerbaycan
Ermenistan
Ukrayna
Belarus
Moldova
Romanya
Bulgaristan
Yunanistan
Sırbistan
Bosna-Hersek
Hırvatistan
Slovenya
Macaristan
Slovakya
Çekya
Polonya
Almanya
Fransa
İtalya
İspanya
Portekiz
İrlanda
İzlanda
İsveç
Finlandiya
Norveç
Danimarka
Hollanda
Belçika
İsviçre
Lihtenştayn
Kosova
Kuzey Makedonya
Arnavutluk
Karadağ
Güney Kıbrıs Rum Yönetimi
Kuzey Kıbrıs Türk Cumhuriyeti
30° 20° 10° 0° 10° 20° 30° 40° 50° 60° 70° 80° 90° 100° 110° 120° 130° 140° 150° 160° 170° 180°
40° 30° 20° 10°
50° 40° 30° 20° 10°
30° 40° 50° 60° 70° 80° 90° 100° 110° 120°