

Daphne Kariyer online eğitim · Dijital yetkinlikler · Yazılım geliştirme

Yazılım Geliştirmeye Giriş

Modül	10
Önerilen süre	Yaklaşık 10–12 saat
Düzye	Başlangıç
Biçim	Online, kendi hızında
Ölçme	Daphne Kariyer ücretsiz testleri

İçindekiler

Modül 1 Yazılım nedir, nasıl geliştirilir?

Modül 2 Algoritma ve problem çözme

Modül 3 Python ile ilk adımlar: değişken, veri tipi, girdi ve çıktı

Modül 4 Karar yapıları ve döngüler

Modül 5 Fonksiyonlar ve modüller

Modül 6 Listeler, sözlükler ve dosyalarla çalışma

Modül 7 Hata ayıklama ve hata yönetimi

Modül 8 Web nasıl çalışır?

Modül 9 Veri tabanına giriş ve SQL'in temeli

Modül 10 Git, takım çalışması, test ve kariyer yolları

Kendini dene cevapları · Sözlük · Sonraki adım

Bu eğitim hakkında

Bu eğitim, hiç kod yazmamış ya da yalnız birkaç deneme yapmış kişiler için hazırlandı. Amaç seni kısa sürede uzman bir yazılımcı yapmak değil; yazılımın nasıl üretildiğini, bir programın nasıl düşünülüp yazıldığını, web sitelerinin ve veri tabanlarının nasıl çalıştığını sağlam bir temelle öğretmek. Eğitimi bitirdiğinde küçük Python programları yazabilir, bir yazılım ekibinin kullandığı terimleri anlayabilir ve hangi alanda ilerlemek istediğine daha bilinçli karar verebilirsin.

Eğitim on modülden oluşur. İlk iki modül yazılımı ve problem çözme mantığını, sonraki beş modül Python ile program yazmayı, son üç modül web, veri tabanı, sürüm kontrolü, test ve kariyer yollarını anlatır. Örnek dil Python 3'tür; web modülünde kısa HTML, CSS ve JavaScript örnekleri de bulunur.

Her modülün sonunda beş soruluk bir "Kendini dene" bölümü var; cevaplar eğitimin sonunda. Kitabın sonunda bir sözlük ve bilgini ölçebileceğin Daphne Kariyer testlerinin listesi yer alır. Eğitim kendi hızında okunmak üzere hazırlandı; eğitmen ya da canlı ders yoktur.

Kod örneklerini yalnız okumakla yetinme: kendin yaz, çalıştır, bir yerini değiştir ve sonucun nasıl değiştiğine bak. Programlama okuyarak değil, yazarak öğrenilir.

Önerilen süre: yaklaşık 10–12 saat (okuma, kod örneklerini çalıştırma ve alıştırma dahil).

Eğitimin sonunda şunları yapabileceksin:

- Yazılım geliştirme sürecinin aşamalarını ve bir yazılım ekibindeki temel rolleri açıklamak.
- Bir problemi adımlara bölüp algoritma ve sözde kod olarak yazmak.
- Python'da değişken, veri tipi, koşul, döngü ve fonksiyon kullanarak küçük programlar yazmak.
- Listeler ve sözlüklerle veri düzenlemek; metin ve CSV dosyalarını okuyup yazmak.
- Hata mesajlarını okuyup hatanın yerini bulmak ve programı beklenmedik girdilere karşı korumak.
- Bir web sayfasının tarayıcıya nasıl geldiğini; HTTP, HTML, CSS, JavaScript, API ve JSON kavramlarını anlatmak.
- Temel SQL sorgularıyla veri tabanından veri okumak, eklemek, güncellemek ve silmek.
- Git ile sürüm kontrolü yapmak, bir ekibin kodu nasıl birlikte geliştirip test ettiğini anlamak ve kendine uygun kariyer yolunu seçmek.

Yazılım nedir, nasıl geliştirilir?

Yazılım, bir bilgisayarın ne yapacağını adım adım tarif eden komutların ve bu komutların kullandığı verilerin bütünüdür. Donanım (işlemci, bellek, disk, ekran) tek başına hiçbir iş yapmaz; ona neyi, hangi sırayla yapacağını yazılım söyler. Telefonundaki banka uygulaması, marketteki kasa programı, belediyenin online randevu sayfası, bir fabrikadaki üretim hattını yöneten sistem ve şu an bu metni gösteren program birer yazılımdır.

Program, kaynak kod ve programlama dili

- **Program:** Belirli bir işi yapan, çalıştırılabilir komutlar dizisi. Bir yazılım ürünü tek bir programdan ya da birlikte çalışan çok sayıda programdan oluşabilir.
- **Kaynak kod:** Programın insanlar tarafından okunabilen, bir programlama dilinde yazılmış metni.
- **Programlama dili:** İnsanın yazabileceği ve bilgisayarın çalıştırabileceği komutların kurallı dili. Python, JavaScript, Java, C#, Go, Kotlin, Swift, PHP ve C yaygın dillerden birkaçıdır.

Bilgisayarın işlemcisi doğrudan yalnız makine kodunu, yani ikili sayılarla ifade edilen çok basit komutları anlar. Yazdığın kaynak kodun bu biçime çevrilmesi gerekir. Bunun iki temel yolu vardır:

- **Derleme (compile):** Kodun tamamı, çalıştırılmadan önce bir derleyici tarafından makine koduna çevrilir. Ortaya çıkan dosya ayrıca çalıştırılır. C, C++, Go ve Rust derlenen dillere örnektir.
- **Yorumlama (interpret):** Kod, bir yorumlayıcı tarafından çalışma sırasında okunup yürütülür. Python ve JavaScript genellikle bu şekilde çalışır.

Gerçekte sınır bu kadar keskin değildir: Java ve C# önce bir ara koda derlenip sanal makinede çalışır, Python da kodu önce bayt koduna (bytecode) çevirir. Başlangıç için şunu bilmen yeter: yazdığın metni çalıştırılabilir hale getiren bir araç vardır ve yazım kurallarına uymayan kod bu aşamada hata verir.

Yazılım türleri

- **Sistem yazılımı:** İşletim sistemleri (Windows, macOS, Linux, Android, iOS) ve donanım sürücüler. Donanımı yönetir, diğer programlara zemin hazırlar.
- **Masaüstü uygulaması:** Bilgisayara kurulan programlar: ofis programları, muhasebe yazılımları, tasarım araçları.
- **Web uygulaması:** Tarayıcıdan açılan, verisi ve iş mantığının bir kısmı sunucuda çalışan uygulamalar: e-ticaret siteleri, internet bankacılığı, kurum içi portallar, e-Devlet hizmetleri.
- **Mobil uygulama:** Telefon ve tabletler için yazılan uygulamalar.
- **Gömülü yazılım:** Bir cihazın içinde çalışan, çoğu zaman kullanıcının fark etmediği yazılım: çamaşır makinesi, araç içi sistemler, POS cihazı, akıllı sayaç.

Yazılım geliştirme süreci

Bir yazılım projesi kod yazmakla başlamaz. İyi işleyen ekiplerde iş kabaca aşağıdaki aşamalardan geçer. Bu aşamaların bütününe yazılım geliştirme yaşam döngüsü (SDLC, Software Development Life Cycle) denir.

1. **İhtiyaç analizi:** Ne yapılacak, kim kullanacak, hangi sorunu çözecek? Kullanıcılarla görüşülür, mevcut iş akışı incelenir, gereksinimler yazılı hale getirilir.

2. **Tasarım:** Sistem hangi parçalardan oluşacak, veri nerede tutulacak, ekranlar nasıl görünecek? Bu aşamada mimari kararlar verilir ve ekran taslakları hazırlanır.
3. **Geliştirme:** Kod yazılır. Büyük işler küçük parçalara bölünür, her parça ayrı geliştirilir.
4. **Test:** Yazılımın gereksinimleri karşılayıp karşılamadığı ve hata içerip içermediği kontrol edilir.
5. **Yayına alma (deployment):** Yazılım, kullanıcıların erişebileceği ortama kurulur: bir sunucuya, uygulama mağazasına ya da kurumun bilgisayarlarına.
6. **Bakım:** Bulunan hatalar düzeltilir, yeni ihtiyaçlara göre özellik eklenir, güvenlik güncellemeleri yapılır. Bir yazılımın ömrünün büyük kısmı bu aşamada geçer.

Şelale ve çevik yaklaşım

Şelale (waterfall) modelinde aşamalar sırayla ve bir kez yapılır: önce bütün analiz biter, sonra bütün tasarım, sonra bütün kod, sonra bütün test. Gereksinimlerin baştan net olduğu ve az değiştiği işlerde (örneğin bir yönetmeliğin birebir uygulandığı bir hesaplama modülünde) işe yarar. Zayıf yanı, kullanıcının ürünü çok geç görmesidir; yanlış anlaşılmış bir ihtiyaç aylar sonra ortaya çıkabilir.

Çevik (agile) yaklaşımda iş kısa döngülere bölünür. Ekip birkaç haftada bir çalışan, küçük ama kullanılabilir bir parça teslim eder, kullanıcıdan geri bildirim alır ve planı günceller. Böylece yanlış yöne gidildiği erken fark edilir.

Çevik yaklaşımın en yaygın uygulamalarından biri **Scrum**'dir. İş genellikle bir ile dört hafta süren **sprint**'lere bölünür. Sprint başında hangi işlerin yapılacağı planlanır, ekip her gün kısa bir toplantıda dün ne yaptığını, bugün ne yapacağını ve önünde bir engel olup olmadığını paylaşır, sprint sonunda yapılan iş gösterilir ve süreç değerlendirilir. Bir diğer yaygın yöntem **Kanban**'dir: işler "Yapılacak", "Yapılıyor", "Bitti" gibi sütunlardan oluşan bir panoda kartlar halinde izlenir ve aynı anda yürütülen iş sayısı sınırlanır.

Yazılım ekibindeki roller

- **Front-end geliştirici:** Kullanıcının gördüğü ve etkileştiği kısmı yazar: sayfa düzeni, düğmeler, formlar, menüler. Temel araçları HTML, CSS ve JavaScript'tir; React, Vue ya da Angular gibi kütüphane ve çatılar (framework) sık kullanılır.
- **Back-end geliştirici:** Sunucu tarafını yazar: iş kuralları, veri tabanı işlemleri, kullanıcı girişi, ödeme gibi işlemlerin arka planı, diğer sistemlerle bağlantı. Python, Java, C#, Go, PHP ve sunucuda çalışan JavaScript (Node.js) yaygın seçeneklerdir.
- **Full stack geliştirici:** Hem front-end hem back-end tarafında çalışabilen geliştirici. Küçük ekiplerde ve yeni kurulan şirketlerde sık aranır.
- **Mobil geliştirici:** Android (Kotlin, Java) ya da iOS (Swift) için uygulama yazar.
- **Test (QA) uzmanı:** Yazılımın doğru çalıştığını sistemli biçimde kontrol eder, test senaryoları hazırlar, bulunduğu hataları kayda geçirir. Testleri otomatikleştiren test otomasyon uzmanları kod da yazar.
- **DevOps ve altyapı mühendisi:** Yazılımın sunuculara kurulmasını, izlenmesini ve otomatik yayına alma hatlarını yönetir.
- **İş analisti ve ürün yöneticisi:** Kullanıcının ihtiyacını anlar, gereksinimleri yazar, önceliklerin belirlenmesine yardım eder.
- **UI/UX tasarımcısı:** Ekranların görünüşünü (UI, kullanıcı arayüzü) ve kullanım deneyimini (UX) tasarlar.

Küçük bir şirkette bu rollerin birkaçını aynı kişi üstlenebilir. Büyük bir bankada ise her rol için ayrı ekipler bulunur.

Örnek vaka

Konya'da 25 şubesi olan bir pastane zinciri, şubelerin günlük siparişlerini merkeze telefon ve e-postayla bildiriyor. Siparişler merkezde bir Excel dosyasına elle giriliyor. Eksik ve hatalı girişler yüzünden bazı şubelere fazla, bazılarına eksik ürün gidiyor; üretim planlamacısı her akşam saatlerce liste kontrol ediyor.

Şirket bir sipariş uygulaması yaptırmaya karar verir. Süreç şöyle ilerler:

1. İş analisti şube sorumlularıyla ve üretim planlamacısıyla görüşür. Gereksinimler çıkar: şube her gün saat 16.00'ya kadar ertesi günün siparişini girebilmeli, merkez toplam üretim listesini görebilmeli, saat 16.00'dan sonra sipariş değiştirilememeli.
2. Tasarımcı sipariş ekranının taslağını hazırlar. İki şube sorumlusu taslağı dener ve "ürünler kategoriye göre gruplansın" önerisini verir.
3. Ekip çevik çalışır. İlk iki haftalık sprintte yalnız sipariş girme ekranı ve merkezin toplam listesi yapılır. Front-end geliştirici ekranı, back-end geliştirici sipariş kayıtlarını ve 16.00 kuralını yazar.
4. Test uzmanı "16.00'dan bir dakika sonra sipariş girmeye çalış", "aynı ürünü iki kez ekle", "eksi miktar gir" gibi senaryoları dener. İki hata bulunur ve düzeltilir.
5. Uygulama önce üç şubede denir, sonra bütün şubelere açılır.
6. Bakım aşamasında yeni istekler gelir: bayram dönemlerinde sipariş saatinin uzatılması, ürün fotoğrafları eklenmesi. Bunlar sonraki sprintlerin listesine eklenir.

Ekibin ilk işinin kod yazmak olmadığına dikkat et. İhtiyacı doğru anlamak, yanlış bir ürünü hızla yazmaktan çok daha değerlidir.

Sık yapılan hatalar

- **İhtiyacı anlamadan koda başlamak.** İşin "zaten belli" olduğu düşünülen projelerde bile kullanıcıyla yapılacak kısa bir görüşme, günlerce yeniden yazmayı önler.
- **Dil seçimini en önemli karar sanmak.** Başlangıçta hangi dille başladığın, programlama mantığını öğrenmekten daha az önemlidir. Bir dili iyi öğrenen kişi ikincisini çok daha hızlı öğrenir.
- **Test ve bakımı unutmak.** Yazılım yayına alınınca bitmez. Kodun okunabilir yazılması, ileride onu düzeltecek kişinin (çoğu zaman yine senin) işini kolaylaştırır.
- **Rolleri kesin duvarlarla ayırmak.** Front-end geliştiricinin sunucunun nasıl çalıştığını, back-end geliştiricinin kullanıcının ekranda ne yaşadığını hiç bilmemesi, ekip içinde iletişimi zorlaştırır.

Kendini dene

1. Derleme ile yorumlama arasındaki temel fark nedir?
2. Yazılım geliştirme yaşam döngüsünün altı aşamasını sırayla say.
3. Şelale modelinin en önemli zayıf yanı nedir?
4. Kullanıcının gördüğü ekranları, düğmeleri ve formları yazan geliştiriciye ne denir?
5. Scrum'da bir ile dört hafta süren çalışma döngüsüne ne ad verilir?

Algoritma ve problem çözme

Algoritma, bir problemi çözmek için izlenen, sırası belli, açık ve sonlu adımlar dizisidir. Yemek tarifi, bir cihazın kurulum kılavuzu ya da bir kurumdaki izin başvuru prosedürü günlük hayattaki algoritmalarıdır. Programlama, bir algoritmayı bilgisayarın çalıştırabileceği bir dilde yazmaktır. Bu yüzden iyi bir programcı, önce iyi bir problem çözücüdür. Dili bilmek, ne yazacağını bilmiyorsan işe yaramaz.

İyi bir algoritmanın özellikleri

- **Açık:** Her adım tek anlama gelir. "Biraz tuz ekle" insan için yeterlidir, bilgisayar için değildir. Bilgisayar "biraz" ne demek, bilmez.
- **Sıralı:** Adımların hangi sırayla yapılacağı bellidir.
- **Sonlu:** Belirli sayıda adımdan sonra biter; sonsuza kadar dönmez.
- **Girdisi ve çıktısı belli:** Ne alır, ne üretir? Fatura hesaplayan bir algoritma ürün listesini alır, ödenecek tutarı üretir.
- **Genel:** Tek bir örnek için değil, aynı türdeki bütün durumlar için çalışır. Yalnız 3 ürünlük sepette doğru sonuç veren bir hesaplama, algoritma değil tesadüftür.

Problem çözme adımları

1. **Problemi kendi cümlele yaz.** Ne isteniyor? Girdi ne, beklenen çıktı ne? Bunu iki üç cümleyle yazamıyorsan problemi henüz anlamamışsındır.
2. **Örnek çöz.** Küçük bir örneği elle, kâğıt üzerinde çöz. Ne yaptığını adım adım not et. Bu notlar algoritmanın ilk taslağıdır.
3. **Parçala.** Büyük problemi küçük alt problemlere böl. Her parçanın ayrı ayrı çözülebilir olmasına çalış.
4. **Algoritmayı yaz.** Önce düz Türkçe ya da sözde kodla, sonra programlama diliyle.
5. **Uç durumları düşün.** Liste boşsa ne olacak? Sayı sıfır ya da negatifse? Kullanıcı sayı yerine harf girerse?
6. **Dene ve düzelt.** Farklı girdilerle çalıştır, beklenen sonuçla gerçekleşen sonucu karşılaştır, fark varsa nedenini bul.

Parçalama (decomposition)

Büyük problemler tek seferde çözülmez. Bir e-ticaret mağazasında "sipariş al" işi ilk bakışta tek bir iş gibi görünür. Parçalarına ayırınca şu alt işler çıkar:

1. Sepetteki ürünlerin stokta olup olmadığını kontrol et.
2. Ürün fiyatlarını topla, varsa indirim kodunu uygula.
3. Kargo ücretini hesapla.
4. Ödemeyi al ve sonucunu kontrol et.
5. Ödeme başarılıysa stoktan düş, sipariş kaydını oluştur.
6. Müşteriye onay e-postası gönder, depoya hazırlama talebi ilet.

Her parça artık küçük ve anlaşılır bir problemdir. Modül 5'te göreceğin fonksiyonlar, bu parçaların koddaki karşılığıdır.

Sözde kod (pseudocode)

Sözde kod, belirli bir programlama dilinin kurallarına bağlı kalmadan, algoritmayı kod benzeri sade cümlelerle yazmaktır. Amaç düşünceyi netleştirmek ve başkasıyla paylaşabilmektir. Örnek: bir e-ticaret mağazası, sepet tutarı 500 TL ve üzerindeyse kargoyu ücretsiz yapıyor, altındaysa 60 TL kargo ücreti ekliyor.

```
BAŞLA
  sepet_tutarını al
  EĞER sepet_tutarı 500 veya daha büyükse
    kargo = 0
  DEĞİLSE
    kargo = 60
  ödenecek = sepet_tutarı + kargo
  ödeneceği göster
BİTİR
```

Aynı algoritma Python'da şöyle yazılır. Kodun ayrıntısını Modül 3 ve 4'te öğreneceksin; şimdilik sözde kodla benzerliğine bak:

```
# Sepet tutarına göre kargo ücretini hesaplar
sepet_tutari = 420
if sepet_tutari >= 500:
    kargo = 0
else:
    kargo = 60
odenecek = sepet_tutari + kargo
print("Ödenecek tutar:", odenecek)
```

Bu kod çalıştırıldığında ekrana "Ödenecek tutar: 480" yazar. Sözde kod ile Python kodu arasındaki benzerlik tesadüf değildir; Python okunabilir olacak şekilde tasarlanmış bir dildir.

Akış şeması

Akış şeması (flowchart) algoritmayı şekillerle gösterir. Başlangıç ve bitiş için oval, işlem için dikdörtgen, karar için eşkenar dörtgen (baklava dilimi), girdi ve çıktı için paralelkenar kullanılır; oklar akışın yönünü gösterir.

Akış şeması özellikle yazılım bilmeyen kişilerle iş akışı konuşurken yararlıdır. Bir muhasebe uzmanı Python bilmez ama "fatura tutarı 10.000 TL'nin üstündeyse müdür onayına git, değilse doğrudan ödeme listesine ekle" kutusunu okuyup "aslında tedarikçi yeniyse tutara bakmadan onay gerekiyor" diye eksik kuralı fark edebilir.

Algoritmanın üç temel yapı taşı

Hangi programı yazarsan yaz, algoritmalar üç temel yapıdan oluşur:

- **Sıra (sequence):** Adımlar yukarıdan aşağı, yazıldığı sırayla çalışır.
- **Seçim (selection):** Bir koşula göre farklı yol izlenir. "Stok sıfırsa siparişi beklet, değilse onayla."
- **Tekrar (iteration):** Bir adım grubu belirli sayıda ya da bir koşul sağlanana kadar tekrarlanır. "Listedeki her müşteri için hatırlatma mesajı hazırla."

Modül 4'te bu yapıların Python karşılıklarını öğreneceksin.

Verimlilik: aynı sonuca farklı yollar

Aynı problemi çözen birden fazla algoritma olabilir ve hepsi aynı hızda değildir. Bir ürün kodunu bir listede aradığını düşün. En basit yol, listenin başından başlayıp tek tek bakmaktır. Buna doğrusal arama denir. 10.000 kayıtlık bir listede doğrusal arama, en kötü durumda 10.000 kayda bakar. Liste sıralıysa daha akıllı bir yol vardır: ortadaki kayda bak, aradığın ondan önce mi sonra mı karar ver ve listenin yarısını at; sonra kalan yarının ortasına bak. Bu yöntemle ikili arama (binary search) denir ve 10.000 kayıtlık sıralı bir listede en fazla 14 adımda sonuca ulaşır.

Başlangıç düzeyinde bu konuda uzmanlaşman gerekmez, ama şunu bil: veri küçükken her algoritma hızlı görünür, veri büyüdükçe algoritma seçimi belirleyici olur. Yazılımcılar bu konuyu "algoritma karmaşıklığı" başlığı altında ve Büyük O gösterimiyle (Big O notation) inceler.

Uygulama

İstanbul'da 60 kişilik bir muhasebe bürosu, personelin haftalık ücretini fazla mesaiyle birlikte hesaplayan küçük bir program istiyor. Büronun kullandığı örnek kural şu: haftada 45 saatin üstündeki her saat fazla mesaidir ve saatlik ücretin 1,5 katı ödenir. Algoritmayı adım adım kuralım.

Adım 1, problemi yaz. Girdiler: haftalık çalışılan saat ve saatlik ücret. Çıktı: haftalık toplam ücret.

Adım 2, elle çöz. Çalışan 50 saat çalışmış, saatlik ücreti 200 TL olsun. Normal kısım $45 \times 200 = 9.000$ TL. Fazla mesai 5 saat, yani $5 \times 200 \times 1,5 = 1.500$ TL. Toplam 10.500 TL. Bir başka çalışan 40 saat çalışmışsa $40 \times 200 = 8.000$ TL alır ve fazla mesaisi yoktur.

Adım 3, uç durumları düşün. Tam 45 saat çalışan fazla mesai alır mı? Kurala göre almaz, çünkü "45'in üstü" deniyor. Negatif saat girilirse? Bu bir veri giriş hatasıdır, hesaplama yapılmamalı. Sıfır saat? Sonuç sıfır olmalı.

Adım 4, sözde kodu yaz:

```
BAŞLA
  saat ve ücreti al
  EĞER saat 0'dan küçükse "Geçersiz saat" göster ve BİTİR
  EĞER saat 45'ten büyükse
    normal = 45 × ücret
    fazla = (saat - 45) × ücret × 1,5
  DEĞİLSE
    normal = saat × ücret
    fazla = 0
  toplam = normal + fazla
  toplamı göster
BİTİR
```

Adım 5, kodu yazmadan önce deneme listesini hazırla. Her satırda girdi ve beklenen sonuç var:

- 40 saat, 200 TL: 8.000 TL
- 45 saat, 200 TL: 9.000 TL
- 50 saat, 200 TL: 10.500 TL
- 0 saat, 200 TL: 0 TL
- Eksi 3 saat: "Geçersiz saat" uyarısı

Bu listeyi kodu yazmadan önce hazırlamak, Modül 10'da göreceğin test yazma alışkanlığının ilk adımıdır. Bu algoritmayı Modül 4'ü bitirdikten sonra Python'a çevirmeyi dene.

Sık yapılan hatalar

- **Elle örnek çözmeden koda başlamak.** Kâğıt üzerinde çözemediğin bir problemi kodla da çözemezsin. Elle çözüm, algoritmanın taslağıdır.
- **Yalnız "mutlu yolu" düşünmek.** Her şeyin beklendiği gibi gittiği durum kolaydır. Boş girdi, sıfır, negatif sayı, çok büyük değerler ve hatalı biçimli veri asıl sorunları çıkarır.
- **Adımları muğlak bırakmak.** "Uygun indirim uygula" bir algoritma adımı değildir. Hangi koşulda yüzde kaç indirim uygulanacağı açıkça yazılmalıdır.
- **Koşulların sınırlarını karıştırmak.** "500 TL'nin üstü" ile "500 TL ve üstü" farklıdır. Tam 500 TL'lik sepet birinde ücretsiz kargo alır, diğerinde almaz. Sınırdaki değerleri mutlaka dene.
- **Bütün problemi tek seferde çözmeye çalışmak.** Parçala, her parçayı ayrı çöz, sonra birleştir.

Kendini dene

1. İyi bir algoritmanın beş özelliğinden üçünü say.
2. Sözde kod ne işe yarar?
3. Akış şemasında karar adımı hangi şekilde gösterilir?
4. Algoritmaların üç temel yapı taşı nelerdir?
5. Sıralı bir listede, her adımda aranan aralığı yarıya indiren arama yöntemine ne ad verilir?

Python ile ilk adımlar: değişken, veri tipi, girdi ve çıktı

Python, okunması kolay yazım kuralları, geniş standart kütüphanesi ve büyük kullanıcı topluluğu nedeniyle başlangıç için sık tercih edilen bir programlama dilidir. Web back-end'inde, veri analizinde, yapay zekâda, test otomasyonunda ve iş otomatikleştiren küçük betiklerde (script) yaygındır. Bu eğitimdeki bütün örnekler Python 3 ile çalışır. Eski bir kaynakta parantezsiz print "merhaba" gibi bir satır görürsen, bu artık desteklenmeyen Python 2 sözdizimidir; o kaynağı kullanma.

Python'u çalıştırmanın yolları

- **Kurulum:** Python'u python.org adresinden indirip kurabilirsin. Windows'ta kurulumun ilk ekranında Python'u PATH'e ekleme seçeneğini işaretle.
- **Etkileşimli kabuk:** Terminale python3 (Windows'ta python ya da py) yazınca açılır; yazdığın her satır anında çalışır. Küçük denemeler için idealdir.
- **Dosyadan çalıştırma:** Kodunu uzantısı .py olan bir dosyaya, örneğin merhaba.py dosyasına yazar, terminalde python3 merhaba.py komutuyla çalıştırırsın.
- **Kod düzenleyici:** Visual Studio Code ve PyCharm yaygın düzenleyicilerdir; Python ile gelen IDLE de başlangıç için yeterlidir. Kurulum istemezsen Google Colab gibi tarayıcıda çalışan ortamları kullanabilirsin.

İlk program

```
# Ekrana iki satır yazdırır
print("Merhaba, dünya")
print("Python öğreniyorum.")
```

print() bir fonksiyondur: parantez içine verdiğin değeri ekrana yazar. Tırnak içindeki ifadeler metindir. Satır başındaki diyez (#) işareti yorum başlatır; Python yorumları çalıştırmaz. Yorumlar kodu okuyan insanlar içindir.

Python büyük ve küçük harfe duyarlıdır. print yerine Print yazarsan program "NameError: name 'Print' is not defined" hatası verir. Hata mesajlarını okumayı Modül 7'de öğreneceksin.

Değişkenler

Değişken, bir değeri bellekte saklamak için kullandığın isimdir. Üzerinde etiket olan bir kutu gibi düşünebilirsin: kutunun içine bir değer koyarsın, sonra o değere etiketin adıyla ulaşırsın. Python'da eşittir işareti (=) "eşittir" değil, "sağdaki değeri soldaki isme ata" anlamına gelir.

```
# Değişken tanımlama ve kullanma
musteri_adi = "Elif"
siparis_adi = 3
birim_fiyat = 149.90
toplam = round(siparis_adi * birim_fiyat, 2)
print(musteri_adi, "toplam:", toplam)
```

Bu kod ekrana "Elif toplam: 449.7" yazar. round() fonksiyonu sayıyı istediğin basamağa yuvarlar. Neden gerektiğini bu modülün sonundaki hatalar listesinde göreceksin.

Değişken adı verirken şu kurallara uyulur:

- Harf ya da alt çizgiyle başlar; rakamla başlayamaz. adet2 geçerlidir, 2adet geçersizdir.
- Boşluk içeremez. Kelimeleri alt çizgiyle ayırırsın: siparis_adedi.
- Büyük ve küçük harf farklıdır. Adet ile adet iki ayrı değişkendir.
- Python'un kendi anahtar kelimeleri (if, for, while, def, class, return, True, False, None gibi) değişken adı olamaz.

Python'da değişken adları alışkanlık olarak küçük harfle ve kelimeler alt çizgiyle ayrılarak yazılır. Değeri hiç değişmeyecek sabitler ise büyük harfle yazılır: KDV_ORANI gibi. Python 3 değişken adlarında Türkçe karakterlere izin verir, ama çoğu ekip sorun çıkmaması için ASCII karakterler (ç yerine c, ş yerine s) kullanır; bu eğitimde de öyle yaptık.

Temel veri tipleri

Her değerın bir tipi vardır ve tip, o değerle neler yapabileceğini belirler.

- **int (tam sayı):** 42, -7, 0 gibi ondalık kısmı olmayan sayılar. Stok adedi, kişi sayısı, yıl.
- **float (ondalıklı sayı):** 3.14, 149.90 gibi sayılar. Python'da ondalık ayırıcı virgül değil noktadır.
- **str (metin, string):** Tek ya da çift tırnak arasındaki karakter dizisi. "Ankara", 'TS-104', "05321234567". Telefon numarası ve posta kodu gibi değerler, baştaki sıfır kaybolmasın diye metin olarak tutulur.
- **bool (mantıksal):** Yalnız iki değeri vardır: True (doğru) ve False (yanlış). "Ödeme alındı mı?", "Müşteri aktif mi?" gibi sorulara cevap tutar.
- **None:** "Değer yok" anlamına gelen özel bir değerdir. Henüz bilinmeyen ya da girilmemiş bir bilgi için kullanılır.

Bir değerin tipini type() fonksiyonuyla görebilirsin:

```
# Veri tiplerini görmek için type() kullanılır
print(type(42))          # <class 'int'>
print(type(3.75))        # <class 'float'>
print(type("Ankara"))    # <class 'str'>
print(type(True))        # <class 'bool'>
print(type(None))        # <class 'NoneType'>
```

Aritmetik işlemler

```
# Aritmetik işlemler
print(7 + 3)             # toplama: 10
print(7 - 3)             # çıkarma: 4
print(7 * 3)             # çarpma: 21
print(7 / 2)             # bölme, sonuç her zaman float: 3.5
print(7 // 2)            # tam bölme, ondalık kısım atılır: 3
print(7 % 2)             # bölümden kalan: 1
print(2 ** 10)           # üs alma: 1024
```

Tam bölme ve kalan işlemleri iş hayatında beklediğinden sık kullanılır. 135 dakikalık bir işin kaç saat kaç dakika olduğunu bulmak için $135 // 60$ işlemi 2 saati, $135 \% 60$ işlemi 15 dakikayı verir. 250 ürün 12'li kolilere konacaksa $250 // 12$ tam koli sayısını (20), $250 \% 12$ açıkta kalan ürün sayısını (10) verir. Bir sayının çift olup olmadığını anlamak için de kalan işlemi kullanılır: sayı $\% 2$ sonucu 0 ise sayı çifttir.

İşlem önceliği matematikteki gibidir. Emin olmadığında parantez kullan.

Metinlerle çalışmak

Metinler programlarda en çok uğraşılacak veri tipidir: müşteri adları, adresler, ürün açıklamaları, e-posta içerikleri. Python'da metinler için hazır işlemler (metot) sunar. Bir metodu kullanmak için değişkenden sonra nokta koyup metodun adını yazarsın.

```
# Metin işlemleri ve f-string
ad = "  ayşe yılmaz "
temiz = ad.strip().title() # baştaki/sondaki boşluğu siler, kelime başlarını büyütür
print(temiz)               # Ayşe Yılmaz
print(len(temiz))          # karakter sayısı: 11
sehir = "İzmir"
print(f"{temiz}, {sehir} şubesiinden sipariş verdi.")
print("05xx".replace("xx", "32")) # metin içinde değiştirme: 0532
```

Bunlara ek olarak upper() bütün harfleri büyük, lower() küçük yapar. strip() özellikle formlardan gelen verilerde gereklidir; kullanıcılar farkında olmadan boşluk bırakır. len() bir metod değil fonksiyondur; len(metin) biçiminde yazılır.

Metinleri birleştirmek için artı işareti kullanılabilir, ama değişkenleri metnin içine yerleştirmenin en okunaklı yolu f-string'tir. Tırnaktan önce f harfi yazarsın, değişkenleri süslü parantez içine koyarsın. f-string içinde biçimlendirme de yapabilirsin: {toplam:.2f} ifadesi sayıyı iki ondalık basamakla yazar.

Türkçe metinlerde dikkat etmen gereken bir durum var. Python'un upper() ve lower() metotları dile özgü kuralları bilmez. "istanbul".upper() sonucu "İSTANBUL" değil "ISTANBUL" olur; "IŞIK".lower() sonucu da "ışık" değil "işik" olur. Türkçe ad ve adresleri büyük harfe çevirirken bu yüzden dikkatli ol; gerekiyorsa i ve ı harflerini önce kendin dönüştür.

Girdi almak: input()

input() fonksiyonu kullanıcıdan klavyeyle bilgi alır. Parantez içine yazdığın metin, kullanıcıya gösterilen sorudur.

```
# Kullanıcıdan girdi alıp sayıya çevirmek
ad = input("Adın nedir? ")
yas = int(input("Kaç yaşındasın? "))
print(f"Merhaba {ad}, 5 yıl sonra {yas + 5} yaşında olacaksın.")
```

Burada çok önemli bir kural var: input() her zaman metin (str) döndürür. Kullanıcı 30 yazsa bile Python bunu "30" metni olarak alır. Metinle sayı toplanamaz; int() ile çevirmeyi unutursan "TypeError: can only concatenate str (not 'int') to str" hatası alırsın. Bu hata, yeni başlayanların en sık karşılaştığı hatalardan biridir.

Tip dönüşümleri

- **int("25"):** Metni tam sayıya çevirir. int("3.5") ya da int("yirmi") hata verir.

- **float("3.5")**: Metni ondalıklı sayıya çevirir.
- **str(100)**: Sayıyı metne çevirir.
- **int(3.9)**: Ondalıklı sayının ondalık kısmını atar ve 3 verir; yuvarlamaz. Yuvarlamak için round() kullanılır.

Türkiye'de kullanıcılar ondalık sayıları genellikle virgülle yazar. float("12,50") hata verir, çünkü Python virgülden ondalık ayırıcı olarak tanımaz. Çözüm, dönüştürmeden önce virgülden noktaya çevirmektir: float(metin.replace(",", ".")).

Uygulama

Ankara'da bir kırtasiye toptancısı, satış temsilcileri için basit bir fiyat hesaplayıcı istiyor: ürün adı, adet ve birim fiyat girilecek; ara toplam, KDV ve genel toplam yazılacak. KDV oranı programın başında sabit olarak tanımlandı; örnekteki yüzde 20 yerine gerçek işte güncel ve ürüne uygun oranı kontrol et.

```
# Basit fiyat hesaplayıcı
KDV_ORANI = 0.20 # örnek oran; güncel oranı kontrol et
urun = input("Ürün adı: ")
adet = int(input("Adet: "))
fiyat_metni = input("Birim fiyat (TL): ")
birim_fiyat = float(fiyat_metni.replace(",", ".")) # 12,50 da kabul edilir
ara_toplam = adet * birim_fiyat
kdv = ara_toplam * KDV_ORANI
genel_toplam = ara_toplam + kdv
print(f"{urun} x {adet}")
print(f"Ara toplam: {ara_toplam:.2f} TL")
print(f"KDV: {kdv:.2f} TL")
print(f"Genel toplam: {genel_toplam:.2f} TL")
```

Kullanıcı "A4 fotokopi kâğıdı", 12 ve "149,90" girerse program ara toplamı 1798.80 TL, KDV'yi 359.76 TL ve genel toplamı 2158.56 TL olarak yazar. Programı kendin çalıştır ve şunları dene:

1. Birim fiyatı noktalı ve virgüllü olarak ayrı ayrı gir; ikisinin de çalıştığını gör.
2. Adet yerine "on iki" yaz. Program hata verir. Neden olduğunu açıklamaya çalış; çözümünü Modül 7'de öğreneceksin.
3. Programa bir iskonto oranı girdisi ekle ve iskontoyu KDV'den önce uygula.

Sık yapılan hatalar

- **input() değerini sayıya çevirmemek.** Kullanıcıdan gelen her şey metindir. Hesap yapacaksan int() ya da float() kullan.
- **Ondalıkta virgül kullanmak.** Kodun içinde 149,90 yazarsan Python bunu iki ayrı sayı (149 ve 90) olarak görür. Kodda her zaman nokta kullan, kullanıcı girdisinde virgülden noktaya çevir.
- **Tırnakları kapatmamak ya da karıştırmak.** "Merhaba" gibi açılışı ve kapanışı farklı tırnaklar SyntaxError verir. İçinde kesme işareti olan metinleri çift tırnakla yaz: "İzmir'e gönderildi".
- **Ondalıklı sayılarda kesinlik beklemek.** Bilgisayarlar ondalıklı sayıları ikili sistemde tuttuğu için 0.1 + 0.2 işleminin sonucu 0.30000000000000004 çıkar. Ekranda gösterirken yuvarla; para gibi kuruşu kuruşuna doğru olması gereken hesaplarda Python'un decimal modülünü kullanmak daha güvenlidir.
- **Türkçe büyük-küçük harf dönüşümüne güvenmek.** upper() ve lower() i, İ, ı, I harflerini Türkçe kurallara göre çevirmez.

- **Anlamsız değişken adları.** a, b, x1 gibi adlar yazarken kolaydır, okurken zordur.

Kendini dene

1. input() fonksiyonu kullanıcıdan aldığı değeri hangi veri tipinde döndürür?
2. 17 // 5 ve 17 % 5 işlemlerinin sonucu nedir?
3. Şunlardan hangisi geçerli bir değişken adıdır: 2adet, adet_2, adet 2?
4. Bir ondalıklı sayıyı f-string içinde iki ondalık basamakla yazdırmak için ne yazarsın?
5. Kullanıcı birim fiyatı "12,50" biçiminde girerse float() neden hata verir, nasıl çözersin?

Karar yapıları ve döngüler

Şimdiye kadar yazdığın programlar yukarıdan aşağı, her satırı bir kez çalıştırdı. Gerçek programlar ise duruma göre farklı yollar izler ve aynı işi birçok kayıt için tekrarlar. Modül 2'de gördüğün seçim ve tekrar yapılarının Python karşılıkları bu modülün konusudur: if, for ve while.

Karşılaştırma ve mantık işlemleri

Karar vermek için önce bir soru sorman gerekir. Python'da sorular karşılaştırma işlemleriyle sorulur ve cevap her zaman True ya da False olur.

```
# Karşılaştırma sonuçları True ya da False döner
stok = 12
print(stok > 0)          # büyük mü: True
print(stok == 0)         # eşit mi: False
print(stok != 12)        # eşit değil mi: False
print(stok <= 12)        # küçük ya da eşit mi: True
uye = True
kampanya_aktif = False
print(uye and kampanya_aktif) # ikisi de doğru mu: False
print(uye or kampanya_aktif)  # en az biri doğru mu: True
print(not kampanya_aktif)     # tersi: True
```

Eşitlik kontrolü için iki eşittir (==) kullanılır. Tek eşittir atama yapar, karşılaştırma yapmaz. Bu ikisini karıştırmak çok sık yapılan bir hatadır.

Mantık işlemlerinden and, iki koşulun da doğru olmasını ister; or, en az birinin doğru olmasını yeterli bulur; not, koşulun tersini alır. "Üyeyse ve sepet 300 TL'nin üstündeyse indirim uygula" cümlesi doğrudan and ile yazılır.

if, elif ve else

```
# Yıllık alışveriş tutarına göre müşteri seviyesi
yillik_tutar = 27500
if yillik_tutar >= 50000:
    seviye = "Altın"
elif yillik_tutar >= 20000:
    seviye = "Gümüş"
elif yillik_tutar >= 5000:
    seviye = "Bronz"
else:
    seviye = "Standart"
print("Müşteri seviyesi:", seviye)
```

Bu kod "Müşteri seviyesi: Gümüş" yazar. Yapıyı inceleyelim:

- if satırı bir koşulla başlar ve iki nokta üst üste (:) ile biter.
- Koşul doğruysa çalışacak satırlar içeriden, yani girintili yazılır. Python'da bir bloğun nereye ait olduğunu girinti belirler. Alışkanlık olarak 4 boşluk kullanılır.

- elif ("else if" kısaltması) önceki koşullar yanlışsa denenecek yeni bir koşuldur. İstedikçe kadar elif yazabilirsin.
- else, hiçbir koşul doğru değilse çalışır. Kullanmak zorunlu değildir.
- Python koşulları yukarıdan aşağı dener ve ilk doğru olanın bloğunu çalıştırır, gerisine bakmaz.

Son madde çok önemlidir. Koşulların sırasını ters yazarsan, yani ilk sıraya yıllık_tutar >= 5000 koşulunu koyarsan, 60.000 TL alışveriş yapan müşteri de ilk koşulu sağladığı için "Bronz" olur. Aralık kontrollerinde en dar ya da en yüksek koşul önce gelir.

for döngüsü

for döngüsü, bir dizi değerin her biri için aynı işi yapar. "Listedeki her şube için", "1'den 12'ye kadar her ay için" gibi cümlelerin karşılığıdır.

```
# range ile 1'den 5'e kadar sayılar
for ay in range(1, 6):
    print("Ay", ay, "raporu hazırlanıyor")
```

range(1, 6) fonksiyonu 1, 2, 3, 4, 5 sayılarını üretir. Bitiş değeri dahil değildir; bu kural başlangıçta kafa karıştırır ama Python'un her yerinde aynıdır. range(5) yazarsan 0'dan başlar ve 0, 1, 2, 3, 4 üretir. Üçüncü bir değer adım büyüklüğüdür: range(0, 100, 10) onar onar artar.

for döngüsü yalnız sayılarla değil, listelerle ve metinlerle de çalışır. Programlamada en sık kullanılan kalıplardan biri biriktirmedir: bir değişkeni sıfırla başlatır, döngünün her adımında üzerine eklersin.

```
# Bir haftalık satışların toplamı ve ortalaması
gunluk_satislar = [12500, 9800, 14300, 11000, 16750, 21400, 8900]
toplam = 0
for satis in gunluk_satislar:
    toplam = toplam + satis
ortalama = toplam / len(gunluk_satislar)
print("Toplam:", toplam)
print(f"Ortalama: {ortalama:.0f}")
```

Köşeli parantez içindeki değerler bir listedir; listeleri Modül 6'da ayrıntılı göreceksin. Aynı toplamı Python'un hazır sum() fonksiyonuyla tek satırda da bulabilirsin: sum(gunluk_satislar).

while döngüsü

while döngüsü, bir koşul doğru olduğu sürece çalışır. Kaç kez tekrar edileceğini baştan bilmediğin durumlar için uygundur.

```
# Mevcut stok kaç gün yeter?
stok = 500
gunluk_satis = 70
gun = 0
while stok >= gunluk_satis:
    stok = stok - gunluk_satis
    gun = gun + 1
print(gun, "gün yeter, kalan stok:", stok)
```

Bu kod "7 gün yeter, kalan stok: 10" yazar. while döngüsünde koşuldaki değişkenin döngü içinde değişmesi şarttır. stok değişkenini azaltan satırı silersen koşul hep doğru kalır ve program hiç bitmez. Buna sonsuz döngü denir. Terminalde sonsuz döngüye giren bir programı Ctrl+C tuşlarıyla durdurabilirsiniz.

while döngüsünün en yaygın kullanımlarından biri, kullanıcı geçerli bir değer girene kadar soruyu tekrarlamaktır:

```
# Geçerli bir adet girilene kadar tekrar sorar
adet = 0
while adet <= 0:
    metin = input("Adet (pozitif tam sayı): ")
    if metin.isdigit():
        adet = int(metin)
    else:
        print("Lütfen yalnız rakam gir.")
print("Kaydedilen adet:", adet)
```

isdigit() metodu, metin yalnız rakamlardan oluşuyorsa True döner. 0 girilirse koşul sağlanmadığı için soru tekrarlanır.

break ve continue

- **break:** Döngüyü hemen bitirir.
- **continue:** Döngünün o adımını atlayıp bir sonrakine geçer.

```
# İptal edilenleri atla, ilk hatalı kayıta dur
siparisler = [250, -1, 400, 180, -99, 300]
for tutar in siparisler:
    if tutar == -1:
        continue # -1: iptal edilmiş sipariş, atla
    if tutar == -99:
        print("Hatalı kayıt bulundu, işlem durdu.")
        break
    print("İşlenen tutar:", tutar)
```

Bu kod 250, 400 ve 180 tutarlarını işler, -1'i atlar, -99'u görünce durur; 300 hiç işlenmez. Döngüleri iç içe de kullanabilirsin: her şube için, o şubenin her ürünü için gibi. İç içe döngülerde break yalnız içinde bulunduğu döngüyü bitirir.

Uygulama

Eskişehir'de küçük bir market, kasada kullanılacak basit bir toplama programı istiyor. Kasiyer ürün fiyatlarını tek tek girecek, boş bırakıp Enter'a bastığında satış bitecek. Program ürün sayısını ve ödenecek tutarı gösterecek; 1.000 TL ve üzeri alışverişte yüzde 5 indirim uygulanacak.

Önce algoritmayı düşün: Kaç ürün girileceğini bilmiyoruz, bu yüzden while kullanacağız. Her adımda bir fiyat okuyup toplama ekleyeceğiz. Boş girişte döngüden çıkacağız. Döngüden sonra indirim kuralını uygulayacağız.

```
# Boş satır girilene kadar fiyat okuyan basit kasa
toplam = 0
urun_sayisi = 0
```

```
while True:
    giris = input("Fiyat (bitirmek için Enter): ")
    if giris == "":
        break
    toplam += float(giris.replace(",", "."))
    urun_sayisi += 1
if toplam >= 1000:
    toplam = toplam * 0.95 # 1000 TL ve üstüne yüzde 5 indirim
print(f"{urun_sayisi} ürün, ödenecek: {toplam:.2f} TL")
```

while True ifadesi, koşulu hep doğru olan bir döngü kurar; döngüden çıkışı içerideki break sağlar. Bu kalıp "ne zaman biteceği içeride belli olacak" durumlar için yaygındır. toplam += 5 yazımı, toplam = toplam + 5 ifadesinin kısaltmasıdır.

Programı geliştirmek için şunları dene:

1. Negatif fiyat girilirse uyarı verip o girişi saymayan bir kontrol ekle (continue işine yarar).
2. En pahalı ürünün fiyatını da gösteren bir değişken ekle.
3. Modül 2'deki fazla mesai algoritmasını if ve else kullanarak Python'a çevir ve hazırladığın beş deneme durumuyla kontrol et.

Sık yapılan hatalar

- **Atama ile karşılaştırmayı karıştırmak.** if stok = 0 yazımı hatadır; doğrusu if stok == 0.
- **İki noktayı unutmak.** if, elif, else, for ve while satırları iki nokta üst üste ile biter.
- **Girinti hataları.** Aynı bloktaki satırların girintisi aynı olmalı. Sekme (Tab) ile boşluğu karıştırmak IndentationError verebilir. Düzenleyicini Tab tuşuna basınca 4 boşluk koyacak şekilde ayarla.
- **Sonsuz döngü.** while koşulundaki değişkeni döngü içinde güncellemeyi unutmak.
- **Bir eksik ya da bir fazla döngü.** range(1, 12) 12'yi içermez; 12 aya dolaşmak için range(1, 13) yazmalısın. Bu hataya İngilizcede "off-by-one" denir.
- **Koşul sırasını yanlış kurmak.** Geniş koşul öne gelirse dar koşula hiç sıra gelmez.
- **Biriktirme değişkenini döngünün içinde sıfırlamak.** toplam = 0 satırı döngünün içine girerse her adımda toplam sıfırlanır ve sonuçta yalnız son değer kalır.

Kendini dene

1. Python'da bir kod bloğunun if ifadesine ait olduğunu ne belirler?
2. range(2, 6) hangi sayıları üretir?
3. for ve while döngüleri hangi durumlarda tercih edilir?
4. Döngüde bir adımı atlayıp sonrakine geçmek için hangi komut kullanılır?
5. Müşteri seviyesi örneğinde yıllık_tutar >= 5000 koşulu ilk sıraya yazılıysaydı, 60.000 TL alışveriş yapan müşteri hangi seviyeyi alırdı?

Fonksiyonlar ve modüller

Programların büyüdükçe aynı hesaplamayı birçok yerde yazmaya başladığını fark edersin: KDV hesabı fatura ekranında, raporda ve teklif formunda tekrar tekrar geçer. Oran değiştiğinde hepsini düzeltmen gerekir ve birini mutlaka unutursun. Fonksiyonlar bu sorunu çözer. Yazılımcılar bu ilkeyi "kendini tekrar etme" (DRY, Don't Repeat Yourself) diye özetler.

Fonksiyon tanımlamak

Fonksiyon, bir işi yapan ve bir adla çağrılan kod bloğudur. Şimdiye kadar `print()`, `input()`, `len()` ve `round()` gibi hazır fonksiyonları kullandın. Kendi fonksiyonunu `def` anahtar kelimesiyle tanımlarsın.

```
# KDV dahil fiyatı hesaplayan fonksiyon
def kdv_dahil(tutar, oran=0.20):
    """Tutara KDV ekler ve iki basamağa yuvarlar."""
    return round(tutar * (1 + oran), 2)

print(kdv_dahil(1000))          # 1200.0
print(kdv_dahil(250, 0.10))    # 275.0
print(kdv_dahil(oran=0.01, tutar=500)) # 505.0
```

Parçaları tek tek inceleyelim:

- **def:** Fonksiyon tanımının başladığını söyler.
- **kdv_dahil:** Fonksiyonun adı. Değişken adlarıyla aynı kurallara uyar. Fonksiyonlar bir iş yaptığı için adlarında genellikle bir fiil ya da sonucu anlatan bir ifade bulunur: `hesapla_kargo`, `musteri_bul`, `kdv_dahil`.
- **tutar ve oran:** Parametreler. Fonksiyonun dışarıdan aldığı değerlerin içerideki adlarıdır.
- **oran=0.20:** Varsayılan değer. Fonksiyon çağrılırken oran verilmezse 0.20 kullanılır.
- **Üç tırnak arasındaki açıklama:** Belge metni (docstring). Fonksiyonun ne yaptığını anlatır; düzenleyiciler bu metni ipucu olarak gösterir.
- **return:** Fonksiyonun ürettiği sonucu çağırana geri verir ve fonksiyonu bitirir.

Fonksiyonu çağırırken verdiğin somut değerlere argüman denir. `kdv_dahil(250, 0.10)` çağrısında 250 ve 0.10 argümandır; sırasıyla tutar ve oran parametrelerine atanır. Son satırdaki gibi parametre adını yazarak da argüman verebilirsin; bu durumda sıra önemli olmaz ve kod daha okunaklı olur.

return ile print arasındaki fark

Yeni başlayanların en sık karıştırdığı konulardan biri budur. `print()` değeri ekrana yazar ve iş biter; program o değeri başka bir yerde kullanamaz. `return` ise değeri çağıran koda geri verir; sen onu bir değişkene atayabilir, başka bir hesapta kullanabilir ya da istersen ekrana yazabilirsin.

```
# print ekrana yazar, return değeri geri verir
def yazdiran(a, b):
    print(a + b)

def donduren(a, b):
    return a + b
```

```
sonuc1 = yazdiran(2, 3)    # ekrana 5 yazar
sonuc2 = donduren(2, 3)   # ekrana bir şey yazmaz
print(sonuc1, sonuc2)     # None 5
```

return içermeyen bir fonksiyon None döndürür. Bu yüzden sonuc1 değişkeninde 5 değil None vardır. Genel kural: hesap yapan fonksiyonlar return kullansın, ekrana yazma işini programın ana kısmı yapsın.

Birden fazla değer döndürmek

Bir fonksiyon virgülle ayırarak birden fazla değer döndürebilir. Çağırın taraf bu değerleri aynı sırayla ayrı değişkenlere alır.

```
# Dakikayı saat ve dakikaya çevirir
def sure_cevir(toplam_dakika):
    saat = toplam_dakika // 60
    dakika = toplam_dakika % 60
    return saat, dakika

s, d = sure_cevir(135)
print(f"{s} saat {d} dakika") # 2 saat 15 dakika
```

Kapsam: değişken nerede yaşar?

Bir fonksiyonun içinde tanımlanan değişken yalnız o fonksiyonun içinde vardır; buna yerel (local) değişken denir. Dosyanın en üst seviyesinde tanımlanan değişkenler ise küreseldir (global) ve fonksiyonların içinden okunabilir.

Fonksiyonun ihtiyaç duyduğu değerleri küresel değişkenlerden okumak yerine parametre olarak alması daha iyidir. Böyle bir fonksiyon aynı girdiyle hep aynı sonucu verir ve kolayca test edilir.

İyi fonksiyon yazmanın kuralları

- **Tek iş yapsın.** "Kargo ücretini hesapla" bir iştir. "Kargo ücretini hesapla, faturayı yazdır ve e-posta gönder" üç iştir; üç ayrı fonksiyon olmalıdır.
- **Adı ne yaptığını söylesin.** islem(), fonk2() gibi adlar yerine kargo_ucreti(), musteri_seviyesi() gibi adlar kullan.
- **Girdiyi parametreyle alsın, sonucu return ile versin.**

Modüller ve import

Modül, içinde fonksiyonlar ve değişkenler bulunan bir Python dosyasıdır. Python, standart kütüphane adı verilen ve kurulumla birlikte gelen çok sayıda hazır modül sunar. Bir modülü kullanmak için import ile programına dahil edersin.

```
# Standart kütüphaneden modül kullanmak
import math
from datetime import date

print(math.ceil(7.2))    # yukarı yuvarlar: 8
print(math.sqrt(144))    # karekök: 12.0
```

```
bugun = date.today()
yil_sonu = date(bugun.year, 12, 31)
print("Yıl sonuna kalan gün:", (yil_sonu - bugun).days)
```

import math modülün tamamını alır; içindekilere math.ceil gibi ulaşırsın. from datetime import date yalnız date'i alır. Sık kullanılan standart modüllerden bazıları şunlardır:

- **math:** Matematik işlemleri: yuvarlama, karekök, pi sayısı.
- **random:** Rastgele sayı üretme ve listeden rastgele seçim. Çekiliş, test verisi üretme.
- **datetime:** Tarih ve saat işlemleri: iki tarih arasındaki gün sayısı, teslim tarihi hesabı.
- **csv ve json:** CSV ve JSON biçimindeki verileri okuma ve yazma. Modül 6 ve 8'de kullanacaksın.
- **decimal:** Kuruş hassasiyeti gereken para hesapları.

Kendi modülünü yazmak

Kendi fonksiyonlarını ayrı bir dosyada toplayıp başka dosyalarda kullanabilirsin. Böylece aynı hesaplama birçok programda ortak kullanılır ve tek bir yerden güncellenir.

```
# hesaplar.py dosyası
def kdv_dahil(tutar, oran=0.20):
    return round(tutar * (1 + oran), 2)

def indirimli(tutar, yuzde):
    return round(tutar * (1 - yuzde / 100), 2)

if __name__ == "__main__":
    # Bu kısım yalnız dosya doğrudan çalıştırılınca çalışır
    print(indirimli(800, 15))
```

```
# ana.py dosyası (hesaplar.py ile aynı klasörde)
import hesaplar

fiyat = hesaplar.indirimli(800, 15)    # 680.0
print(hesaplar.kdv_dahil(fiyat))      # 816.0
```

hesaplar.py dosyasının sonundaki kalıbı birçok Python projesinde göreceksin. Dosya doğrudan çalıştırıldığında (python3 hesaplar.py) o bloktaki satırlar çalışır; dosya başka bir dosyadan import edildiğinde çalışmaz. Böylece bir modülün içine onu denemek için küçük kodlar koyabilir, bu kodların modülü kullanan programlarda çalışmasını engellersin.

Paketler, pip ve sanal ortam

Standart kütüphane dışında, başkalarının yazıp paylaştığı paketler de vardır. Python paketleri Python Package Index (PyPI) üzerinde yayımlanır ve pip aracıyla kurulur. Örneğin web istekleri için requests, veri analizi için pandas, Excel dosyaları için openpyxl yaygın paketlerdir.

Her projenin paketlerini ayrı tutmak için sanal ortam (virtual environment) kullanılır. Böylece bir projede kurduğun paket sürümü diğer projeyi bozmaz.

```
# Sanal ortam oluşturup paket kurmak (macOS / Linux)
python3 -m venv .venv
```

```
source .venv/bin/activate
pip install requests
pip freeze > requirements.txt # kurulu paketleri listeye yazar
```

Windows'ta sanal ortamı etkinleştirme komutu `.venv\Scripts\activate` biçimindedir. `requirements.txt` dosyası, projeyi başka bir bilgisayarda kuracak kişinin `pip install -r requirements.txt` komutuyla aynı paketleri kurmasını sağlar.

Yalnız bilinen ve bakımı süren paketleri kur, adını dikkatle yaz: popüler paketlere çok benzeyen adlarla zararlı paketler yayımlanabilir.

Uygulama

Gaziantep'te ev tekstili satan bir e-ticaret mağazası, kargo ücretini desiye göre hesaplıyor. Desi, paketin hacmine göre hesaplanan bir ölçüdür. Örnek tarife şöyle:

- 0 ya da daha küçük desi geçersizdir.
- 3 desiye kadar sabit 70 TL.
- 3'ten büyük 10'a kadar olan kısımda, 3'ün üstündeki her desi için 12 TL eklenir.
- 10 desinin üstünde 154 TL'ye (10 desinin ücreti) her ek desi için 9 TL eklenir.
- Aynı şehir içindeki gönderilerde toplam ücrete yüzde 20 indirim uygulanır.

```
# Örnek tarife göre kargo ücreti
def kargo_ucreti(desi, sehir_ici=False):
    if desi <= 0:
        return None
    if desi <= 3:
        ucret = 70
    elif desi <= 10:
        ucret = 70 + (desi - 3) * 12
    else:
        ucret = 154 + (desi - 10) * 9
    if sehir_ici:
        ucret = ucret * 0.8
    return round(ucret, 2)
for d in [2, 5, 10, 15]:
    print(d, "desi:", kargo_ucreti(d), "TL")
```

Program 2 desi için 70, 5 desi için 94, 10 desi için 154, 15 desi için 199 TL yazar. `kargo_ucreti(5, True)` çağrısı şehir içi indirimiyle 75.2 döner. Geçersiz desi için fonksiyon `None` döndürür; çağırılan kod bu durumu kontrol edip kullanıcıya uyarı gösterebilir.

Tarife tek bir yerde durduğu için fiyatlar değiştiğinde yalnız bu fonksiyon güncellenir; onu kullanan bütün ekranlar doğru ücreti gösterir. Şunları dene:

1. Sepet tutarı 750 TL ve üzerindeyse ücreti sıfır yapan bir `sepet_tutari` parametresi ekle. Varsayılan değeri ne olmalı, düşün.
2. 3 ve 10 gibi sınır değerlerde fonksiyonun doğru çalıştığını kontrol et. Sınırlar hataların en sık saklandığı yerlerdir.

Sık yapılan hatalar

- **Fonksiyonu tanımlayıp çağırmamak.** def ile tanımlanan kod, çağrılana kadar çalışmaz. Tanımdan önce çağırmak da NameError verir.
- **Parantezi unutmak.** print(kdv_dahil) yazarsan fonksiyonun sonucu değil, fonksiyonun kendisiyle ilgili teknik bir bilgi görürsün. Çağırmak için parantez ve argüman gerekir.
- **return yerine print kullanmak.** Sonucu başka bir hesapta kullanacaksan return şarttır.
- **return'den sonra kod yazmak.** return çalıştığında fonksiyon biter; altındaki satırlar çalışmaz.
- **Dosyana standart bir modülün adını vermek.** Kendi dosyana random.py ya da math.py adını verirsen, import random satırı Python'un modülü yerine senin dosyanı yükler ve beklenmedik hatalar çıkar.
- **Her şeyi tek bir fonksiyona yığmak.** Yüz satırlık bir fonksiyonu test etmek ve düzeltmek zordur. Parçala.

Kendini dene

1. Parametre ile argüman arasındaki fark nedir?
2. return içermeyen bir fonksiyon çağrıldığında ne döner?
3. Varsayılan değerli parametre ne işe yarar? Bir örnek ver.
4. Kendi kod dosyana random.py adını vermek neden sorun çıkarır?
5. PyPI'daki üçüncü taraf bir Python paketini kurmak için hangi araç kullanılır?

Listeler, sözlükler ve dosyalarla çalışma

Gerçek işlerde tek tek değerlerle değil, veri gruplarıyla çalışırsın: bir şubenin bütün satışları, bir müşterinin bütün bilgileri, bir depodaki bütün ürünler. Python'da bu tür verileri tutmak için en çok kullanılan iki yapı liste ve sözlüktür. Bu modülde ayrıca dosyalara yazmayı ve dosyalardan okumayı öğreneceksin.

Listeler

Liste, sıralı bir değerler topluluğudur. Köşeli parantezle yazılır, değerler virgülle ayrılır. Bir liste farklı tipte değerler içerebilir ama çoğu zaman aynı türde şeyleri tutar: şehir adları, fiyatlar, sipariş numaraları.

```
# Liste oluşturma ve temel işlemler
sehirler = ["Ankara", "İzmir", "Bursa"]
sehirler.append("Adana")          # sona ekler
print(sehirler[0])                # ilk eleman: Ankara
print(sehirler[-1])               # son eleman: Adana
print(sehirler[1:3])              # dilim: ['İzmir', 'Bursa']
sehirler.remove("Bursa")          # değere göre siler
print(len(sehirler), "şehir")     # 3 şehir
print("İzmir" in sehirler)        # var mı: True
```

Listelerle ilgili temel kurallar:

- **Sıra numarası (indeks) 0'dan başlar.** İlk eleman `sehirler[0]`, ikinci eleman `sehirler[1]`'dir. Eksi indeks sondan sayar: `sehirler[-1]` son elemandır.
- **Dilimleme:** `sehirler[1:3]` ifadesi 1. ve 2. elemanları verir; `range`'de olduğu gibi bitiş dahil değildir.
- **Ekleme ve silme:** `append()` sona ekler, `insert(sıra, deger)` istenen sıraya ekler, `remove(deger)` değeri bulup siler, `pop()` son elemanı çıkarıp döndürür.
- **Sıralama:** `sort()` listeyi yerinde sıralar, `sorted(liste)` sıralanmış yeni bir liste döndürür. Ters sıra için `reverse=True` verilir.

Türkçe metinleri sıralarken dikkat: Python harfleri bilgisayardaki karakter kodlarına göre sıralar, Türk alfabesine göre değil. Bu yüzden "Çorum" listede "Zonguldak"tan sonra gelir, çünkü Ç harfinin kodu Z'den büyüktür. Kullanıcıya alfabetik liste göstereceksen bu durumu hesaba kat.

Bir listeden yeni bir liste üretmenin kısa yolu liste üreticidir (list comprehension):

```
# Liste üretici: bütün fiyatlara yüzde 10 zam
fiyatlar = [100, 250, 80]
yeni_fiyatlar = [round(f * 1.10, 2) for f in fiyatlar]
print(yeni_fiyatlar)                # [110.0, 275.0, 88.0]
pahalilar = [f for f in fiyatlar if f > 90]
print(pahalilar)                    # [100, 250]
```

Listeye benzeyen ama değiştirilemeyen bir yapı daha vardır: demet (tuple). Normal parantezle yazılır: `koordinat = (39.92, 32.85)`. Modül 5'te birden fazla değer döndüren fonksiyon aslında bir demet döndürüyordu.

Sözlükler

Sözlük (dictionary), anahtar ve değer çiftlerinden oluşur. Süslü parantezle yazılır. Listede bir değere sıra numarasıyla ulaşırız; sözlükte ise anlamlı bir anahtarla: `urun["fiyat"]`. Bir müşteri kaydı, bir ürün kartı, bir ayar dosyası sözlükle doğal biçimde ifade edilir.

```
# Sözlük: anahtar-değer çiftleri
urun = {"kod": "TS-104", "ad": "Termos", "fiyat": 349.0, "stok": 18}
print(urun["ad"])           # Termos
urun["stok"] = urun["stok"] - 3 # değeri günceller
urun["renk"] = "lacivert"    # yeni anahtar ekler
print(urun.get("garanti", "yok")) # anahtar yoksa varsayılan: yok
for anahtar, deger in urun.items():
    print(anahtar, ":", deger)
```

Sözlüklerle ilgili temel kurallar:

- **Anahtarlar benzersizdir.** Aynı anahtara yeni bir değer atarsan eskisi silinir.
- **Olmayan anahtara köşeli parantezle ulaşmak hata verir.** `urun["garanti"]` yazarsan `KeyError` alırsın. Anahtarın olup olmadığından emin değilsen `get()` kullan; ikinci argüman, anahtar yoksa dönecek değerdir.
- **Döngüyle dolaşmak:** `items()` anahtar ve değeri birlikte, `keys()` yalnız anahtarları, `values()` yalnız değerleri verir.
- **Silme:** `del urun["renk"]` bir anahtarı ve değerini siler.
- **Kontrol:** `"fiyat"` in `urun` ifadesi anahtarın var olup olmadığını söyler.

Sözlüklerin çok kullanışlı bir kalıbı sayımdır. Bir listede hangi değerden kaç tane olduğunu bulmak için:

```
# Sözlükle sayım: hangi üründen kaç sipariş geldi?
siparisler = ["simit", "poğaç", "simit", "açma", "simit", "poğaç"]
sayac = {}
for urun in siparisler:
    sayac[urun] = sayac.get(urun, 0) + 1
print(sayac) # {'simit': 3, 'poğaç': 2, 'açma': 1}
```

Liste içinde sözlük: küçük bir tablo

Gerçek hayatta verinin çoğu tablo biçimindedir: her satır bir kayıt, her sütun bir alan. Python'da bunu ifade etmenin yaygın yolu sözlüklerden oluşan bir listedir. Her sözlük bir satırdır.

```
# Sözlüklerden oluşan liste: küçük bir tablo gibi
calisanlar = [
    {"ad": "Murat", "birim": "Depo", "kalan_izin": 12},
    {"ad": "Selin", "birim": "Muhasebe", "kalan_izin": 4},
    {"ad": "Kerem", "birim": "Depo", "kalan_izin": 9},
]
depo = [c["ad"] for c in calisanlar if c["birim"] == "Depo"]
print("Depo çalışanları:", depo)
toplam_izin = sum(c["kalan_izin"] for c in calisanlar)
print("Toplam kalan izin günü:", toplam_izin)
```

Dosyalarla çalışmak

Değişkenlerdeki veriler program kapanınca kaybolur; kalıcı veriler dosyaya yazılır. Dosyalar open() fonksiyonuyla açılır. Dosyayı açarken bir kip (mode) belirtirsin:

- **"r"**: Okuma. Varsayılan kiptir. Dosya yoksa hata verir.
- **"w"**: Yazma. Dosya yoksa oluşturur; varsa içeriğini tamamen siler ve baştan yazar.
- **"a"**: Ekleme. Dosyanın sonuna yazar; mevcut içeriği korur. Kayıt (log) dosyaları için uygundur.

Dosyaları with yapısıyla açmak iyi bir alışkanlıktır: with bloğu bittiğinde dosya, bir hata olsa bile otomatik olarak kapanır.

```
# Dosyaya yazma ve dosyadan okuma
with open("notlar.txt", "w", encoding="utf-8") as dosya:
    dosya.write("Toplantı saat 10.00'da\n")
    dosya.write("Şube raporları cuma günü\n")

with open("notlar.txt", "r", encoding="utf-8") as dosya:
    for satir in dosya:
        print(satir.strip())
```

Metindeki \n ifadesi satır sonu demektir. write() kendiliğinden satır sonu eklemeyiz; her satırın sonuna sen eklersin. Okurken her satırın sonunda bu karakter bulunduğu için strip() ile temizlenir.

encoding="utf-8" parametresini her zaman yaz. Belirtmezsen Python işletim sisteminin varsayılan kodlamasını kullanır; bu kodlama sistemden sisteme değişebildiği için Türkçe karakterler bozulabilir ya da dosya başka bir bilgisayarda okunamayabilir.

CSV dosyaları

CSV (virgülle ayrılmış değerler), tablo biçimindeki verileri düz metin olarak saklayan basit ve yaygın bir biçimdir. Excel, muhasebe programları ve birçok web paneli CSV dışı aktarma sunar. Python'un csv modülü bu dosyaları kolayca okur ve yazar.

```
# CSV dosyası oluşturup okumak
import csv

with open("stok.csv", "w", newline="", encoding="utf-8") as f:
    yazici = csv.writer(f)
    yazici.writerow(["urun", "adet"])
    yazici.writerows([["kalem", 120], ["defter", 45], ["silgi", 8]])

with open("stok.csv", encoding="utf-8") as f:
    for satir in csv.DictReader(f):
        if int(satir["adet"]) < 50:
            print("Az kaldı:", satir["urun"])
```

csv.DictReader her satırı, ilk satırdaki başlıkları anahtar olarak kullanan bir sözlük olarak verir. Bu yüzden satir["adet"] yazarak değere ulaşabilirsin. CSV'den okunan bütün değerler metindir; karşılaştırma ya da hesap yapmadan önce int() veya float() ile çevirmen gerekir.

Türkiye'de sık karşılaşılan iki durum var. Birincisi, Türkçe bölge ayarlı Excel, CSV dosyalarını virgül yerine noktalı virgülle ayırarak kaydedebilir; bu durumda csv.reader(f, delimiter=";") biçiminde ayırıcıyı belirtirsin. İkincisi, Excel'den UTF-8 olarak kaydedilen dosyaların başında görünmeyen bir işaret (BOM) bulunabilir; bu işaret ilk başlığın adını bozar. Dosyayı encoding="utf-8-sig" ile açmak bu sorunu çözer.

Uygulama

İzmir'de üç şubesi olan bir pastane, gün sonunda şubelerden gelen satış kayıtlarını tek bir CSV dosyasında topluyor. Muhasebeci şube toplamalarını her akşam elle hesaplıyor. Önce deneme için örnek bir dosya oluşturalım:

```
# Örnek veri: satislar.csv dosyasını oluşturur
import csv

kayitlar = [
    ["sube", "urun", "tutar"],
    ["Karşıyaka", "pasta", "1250"],
    ["Bornova", "kurabiye", "430"],
    ["Karşıyaka", "börek", "380"],
    ["Çiğli", "pasta", "990"],
    ["Bornova", "pasta", "1100"],
]

with open("satislar.csv", "w", newline="", encoding="utf-8") as f:
    csv.writer(f).writerows(kayitlar)
```

Şimdi dosyayı okuyup her şubenin toplamını bir sözlükte biriktiren, sonucu hem ekrana hem de rapor.txt dosyasına yazan programı yazalım:

```
# Şube bazında toplam satış raporu
import csv

toplamlar = {}
with open("satislar.csv", encoding="utf-8") as f:
    for satir in csv.DictReader(f):
        sube = satir["sube"]
        toplamlar[sube] = toplamlar.get(sube, 0) + float(satir["tutar"])

with open("rapor.txt", "w", encoding="utf-8") as rapor:
    for sube, toplam in sorted(toplamlar.items()):
        rapor.write(f"{sube}: {toplam:.2f} TL\n")
    print(f"{sube}: {toplam:.2f} TL")
```

Program Bornova için 1530.00, Karşıyaka için 1630.00 ve Çiğli için 990.00 TL yazar. Sıralamaya dikkat et: Çiğli, Türk alfabesine göre Karşıyaka'dan önce gelmesi gerekirken en sona düştü. Bu, yukarıda anlattığımız karakter kodu sıralamasının somut bir örneğidir.

Geliştirmek için şunları dene:

1. Her şubede hangi üründen ne kadar satıldığını gösteren iç içe bir sözlük kur: {"Bornova": {"pasta": 1100, "kurabiye": 430}}.
2. Rapora bütün şubelerin genel toplamını ve en çok satış yapan şubeyi ekle.

Büyük verilerde bu tür işler için pandas kütüphanesi kullanılır; mantığı aynıdır: satırları oku, grupla, topla.

Sık yapılan hatalar

- **İndeksin 0'dan başladığını unutmak.** Üç elemanlı listede son eleman `liste[2]`'dir; `liste[3]` "IndexError: list index out of range" hatası verir.
- **Olmayan anahtara köşeli parantezle ulaşmak.** `KeyError` almamak için `get()` kullan ya da önce `in` ile kontrol et.
- **"w" kipinin dosyayı sildiğini unutmak.** Mevcut bir dosyaya ekleme yapacaksan "a" kullan. Önemli dosyalarla denemeye başlamadan önce kopyasını al.
- **encoding belirtmemek.** Türkçe karakterli dosyalarda her zaman `encoding="utf-8"` yaz.
- **CSV'den gelen değerleri sayı sanmak.** "45" ile 45 farklıdır; "45" < "5" karşılaştırması metin olarak yapıldığı için True döner.
- **Listeyi kopyaladığını sanmak.** `yeni = eski` yazmak listeyi kopyalamaz; iki ad aynı listeyi gösterir ve birinde yaptığın değişiklik diğesinde de görünür. Kopya için `eski.copy()` kullan.

Kendini dene

1. `sehirler = ["Ankara", "İzmir", "Bursa"]` listesinde `sehirler[-1]` neyi verir?
2. Sözlükte olmayan bir anahtara köşeli parantezle ulaşınca ne olur, güvenli yol nedir?
3. Bir dosyayı "w" ve "a" kipiyle açmak arasındaki fark nedir?
4. Dosya işlemlerinde `with` yapısı ne sağlar?
5. CSV dosyasından okunan "45" değeriyle toplama yapmadan önce ne yapmalısın?

Hata ayıklama ve hata yönetimi

Kod yazan herkes hata yapar; fark, hatayı bulma ve düzeltme hızındadır. Programdaki hataya İngilizcede "bug", hatayı bulup düzeltme işine "debugging" (hata ayıklama) denir. Bu modülde hata türlerini, hata mesajlarını okumayı, hatanın yerini bulma yöntemlerini ve programını beklenmedik durumlara karşı korumayı öğreneceksin.

Üç tür hata

- **Sözdizimi hatası (SyntaxError):** Kod, dilin yazım kurallarına uymuyor. Kapanmamış parantez, unutulmuş iki nokta, yanlış girinti. Python bu hatayı program çalışmaya başlamadan bildirir. Bulması en kolay hata türüdür.
- **Çalışma zamanı hatası (exception, istisna):** Kod yazım olarak doğru ama çalışırken bir sorunla karşılaşılıyor: sıfıra bölme, olmayan bir dosyayı açma, sayıya çevrilemeyen bir metni çevirme. Program o satırda durur ve hata mesajı verir.
- **Mantık hatası:** Program hata vermeden çalışır ama yanlış sonuç üretir. Python bu hatayı bilemez; yazdığın şeyi yapmıştır, istediğin şeyi değil. Bulması en zor hata türüdür ve test yazmanın asıl nedeni budur.

Hata mesajını okumak

Python bir çalışma zamanı hatasıyla karşılaştığında "traceback" adı verilen bir rapor yazar. İlk görüşte karmaşık görünür ama düzenli bir yapısı vardır:

```
Traceback (most recent call last):
  File "fiyat.py", line 3, in <module>
    adet = int(input("Adet: "))
ValueError: invalid literal for int() with base 10: 'on iki'
```

Traceback'i aşağıdan yukarıya oku:

1. **Son satır** hatanın türünü ve kısa açıklamasını verir. Burada tür ValueError, açıklama da "on iki" metninin tam sayıya çevrilemediğidir.
2. **Üstündeki satırlar** hatanın hangi dosyanın hangi satırında oluştuğunu ve o satırdaki kodu gösterir: fiyat.py dosyasının 3. satırı.

Sık karşılaşılabilecek hata türleri ve genellikle anlattıkları şunlardır:

- **NameError:** Tanımlanmamış bir ad kullandın. Çoğu zaman yazım hatasıdır: toplam yerine toplm yazmak gibi.
- **TypeError:** Bir işlemi uygun olmayan tipte bir değerle yapmaya çalıştın: metinle sayıyı toplamak, fonksiyona eksik argüman vermek.
- **ValueError:** Tip doğru ama değer uygun değil: int("on iki").
- **IndexError:** Listede olmayan bir sıra numarasına ulaşmaya çalıştın.
- **KeyError:** Sözlükte olmayan bir anahtara ulaşmaya çalıştın.
- **ZeroDivisionError:** Sıfıra böldün.
- **FileNotFoundError:** Açmaya çalıştığın dosya belirttiğin yerde yok.

Hata ayıklama yöntemleri

1. **Hatayı yeniden üret.** Hangi girdiyle, hangi adımlarla oluştuğunu bul.
2. **Mesajı oku.** Hata türü ve satır numarası çoğu zaman sorunun nerede olduğunu doğrudan söyler.
3. **Ara değerleri yazdır.** Şüphelendiğin yerlere print() ekleyip değişkenlerin o andaki değerine ve tipine bak: print("adet:", adet, type(adet)). Basit ama çok etkili bir yöntemdir.
4. **Sorunu küçült.** Hatayı oluşturan en küçük kod parçasını bul.
5. **Hata ayıklayıcı (debugger) kullan.** Visual Studio Code ve PyCharm gibi düzenleyicilerde bir satıra kesme noktası (breakpoint) koyup programı o satırda durdurabilir, değişkenlerin değerlerini görebilir ve kodu satır satır ilerletebilirsin. Python'da koda breakpoint() satırı eklemek de terminalde yerleşik hata ayıklayıcıyı açar.
6. **Anlat.** Kodu satır satır, yüksek sesle birine ya da bir nesneye anlat ("lastik ördek yöntemi"). Anlatırken çoğu zaman hatayı kendin fark edersin.
7. **Ara.** Hata mesajının son satırını arama motorunda arat, Python'un resmî belgelerine bak. Yapay zekâ araçlarına gizli kod, şifre ya da müşteri verisi yapıştırma; önerilen düzeltmeyi anlamadan kullanma.
8. **Bir seferde tek değişiklik yap.** Aynı anda beş şeyi değiştirirsen hangisinin sorunu çözdüğünü (ya da yenisini yarattığını) bilemezsin.

Bir mantık hatası örneğine bakalım:

```
# Mantık hatası: ortalama yanlış çıkıyor
def ortalama(sayilar):
    toplam = 0
    for s in sayilar:
        toplam = s          # hata: toplam += s olmalı
    return toplam / len(sayilar)

print(ortalama([10, 20, 30])) # 20 beklenir, 10.0 çıkar
```

Program hata vermez ama 20 yerine 10.0 yazar. Döngünün içine print("s:", s, "toplam:", toplam) satırını eklersen toplamın her adımda biriktirilmediğini, yalnız son değeri tuttuğunu görürsün: 10, 20, 30. Son değer 30, 3'e bölününce 10. Düzeltme, toplam = s satırını toplam += s yapmaktır.

try ve except ile hataları yakalamak

Bazı hatalar senin kodundan değil, dış dünyadan kaynaklanır: kullanıcı yanlış değer girer, dosya silinmiştir, internet bağlantısı kopar. Bu durumlarda programın çökmesi yerine durumu kontrollü biçimde ele alması gerekir. Python'da bunun yolu try ve except yapısıdır.

```
# Kullanıcı hatasını yakalamak
metin = input("Kaç kişilik masa? ")
try:
    kisi = int(metin)
    print("Kişi başı hesap:", 1800 / kisi)
except ValueError:
    print("Lütfen sayı gir.")
except ZeroDivisionError:
    print("Kişi sayısı sıfır olamaz.")
```

Python önce try bloğunu çalıştırır. Bir hata oluşursa bloğun geri kalanını atlar ve hatanın türüne uyan except bloğuna geçer. Hata oluşmazsa except blokları hiç çalışmaz. Farklı hata türleri için ayrı except blokları yazarak her duruma uygun mesaj verebilirsin.

Bu yapının iki ek parçası daha vardır:

- **else:** try bloğunda hata oluşmazsa çalışır. Yalnız her şey yolundaysa yapılacak işleri buraya koyarsın.
- **finally:** Hata olsa da olmasa da her durumda çalışır. Bağlantı kapatma, geçici dosya silme gibi temizlik işleri için kullanılır.

```
# else ve finally kullanımı
try:
    dosya = open("ayarlar.txt", encoding="utf-8")
except FileNotFoundError:
    print("Ayar dosyası yok, varsayılan ayarlar kullanılıyor.")
else:
    print(dosya.read())
    dosya.close()
finally:
    print("Ayar kontrolü bitti.")
```

Kendi hatanı fırlatmak: raise

Bazen hata Python'un gözünde hata değildir ama senin iş kuralına göre hatadır. Yüzde 70 iskonto Python için geçerli bir sayıdır; şirketin kuralına göre ise iskonto en fazla yüzde 50 olabilir. Bu durumda raise ile kendi hatanı fırlatırsın:

```
# Geçersiz veride kendi hatanı fırlatmak
def iskonto_uygula(tutar, yuzde):
    if not 0 <= yuzde <= 50:
        raise ValueError("İskonto 0 ile 50 arasında olmalı")
    return tutar * (1 - yuzde / 100)

try:
    print(iskonto_uygula(1000, 70))
except ValueError as hata:
    print("Hata:", hata)
```

Bu kod "Hata: İskonto 0 ile 50 arasında olmalı" yazar. except satırındaki as hata ifadesi, hatanın açıklamasını bir değişkene alır. Geçersiz veriyi sessizce düzeltmek (örneğin 70'i kendiliğinden 50 yapmak) yerine açıkça hata vermek daha güvenlidir; yanlış bir değerın sistemde fark edilmeden dolaşmasını önler.

Kayıt tutmak: logging

Gece çalışan bir rapor betiği ya da sunucu uygulaması için ekrana yazmak işe yaramaz; kimse ekrana bakmaz. Bu tür programlar ne yaptıklarını ve sorunlarını bir kayıt (log) olarak tutar. Python'un logging modülü bu iş içindir.

```
# logging ile zaman damgalı kayıt tutmak
import logging

logging.basicConfig(level=logging.INFO,
```

```
format="%(%asctime)s %(levelname)s %(message)s")
logging.info("Gün sonu raporu başladı")
logging.warning("Bornova şubesinde veri gelmedi")
logging.error("Rapor dosyası yazılamadı")
```

Kayıt seviyeleri önem sırasıyla DEBUG, INFO, WARNING, ERROR ve CRITICAL'dır; hangi seviyeden itibaren kayıt tutulacağını ayarlayabilirsin.

Kayıtlara kişisel veri yazma. TC kimlik numarası, kart numarası, şifre, sağlık bilgisi gibi veriler log dosyalarına girdiğinde, 6698 sayılı Kişisel Verilerin Korunması Kanunu (KVKK) açısından ayrıca korunması gereken yeni bir kopya oluşur ve çoğu zaman bu dosyalar asıl veri tabanı kadar iyi korunmaz. Kayıta kişiyi tanımlamak gerekiyorsa sistemdeki iç numarayı kullan.

Örnek vaka

Mersin'de 35 kişilik bir lojistik firmasında çalışan bir operasyon uzmanı, her sabah teslimat tarihlerini içeren bir dosyayı okuyup geciken gönderileri listeleyen bir Python betiği yazmış. Betik bir sabah çoküyor ve ekip gecikme listesini alamıyor.

Uzman hata ayıklama adımlarını uyguluyor:

1. **Mesajı okuyor.** Traceback'in son satırı bir ValueError gösteriyor: "time data '15.13.2026' does not match format '%d.%m.%Y'". Yani bir tarih, beklenen gün.ay.yıl biçimine uymuyor.
2. **Hatayı yeniden üretiyor ve veriye bakıyor.** Bir kaydın elle yanlış girildiğini (13. ay yok), bir satırda da tarihin boş olduğunu görüyor.
3. **Kural belirliyor.** Tek bir hatalı satır bütün listeyi engellememeli, ama sessizce yok sayılırsa geciken bir gönderi gözden kaçabilir. Karar: hatalı satırlar atlanacak, her biri satır numarasıyla raporlanacak ve elle kontrol edilecek.
4. **Düzeltiliyor ve deniyor.**

```
# Hatalı tarihleri atlayıp hangi satır olduğunu bildiren okuma
from datetime import datetime

satir_lar = ["03.11.2026", "", "15.13.2026", "07.11.2026"]
gecerli = []
for no, metin in enumerate(satir_lar, start=1):
    try:
        tarih = datetime.strptime(metin, "%d.%m.%Y").date()
        gecerli.append(tarih)
    except ValueError:
        print(f"Satır {no} atlandı, kontrol et: '{metin}'")
print(len(gecerli), "geçerli tarih okundu")
```

enumerate() fonksiyonu, döngüde her elemanın yanında sıra numarasını da verir; start=1 ile numaralar 1'den başlar. strptime() metni belirtilen biçime göre tarihe çevirir, uymazsa ValueError verir.

1. **Kökü düşünüyor.** Tarihler elle yazılan bir tablodan geliyor. Uzman, o tabloda tarih sütununa veri doğrulama eklenmesini öneriyor. En iyi hata yönetimi, hatalı verinin sisteme hiç girmemesidir.

Sık yapılan hatalar

- **Mesajı okumadan kodu rastgele değiştirmek.** Hata mesajı, sorunun yerini çoğu zaman doğrudan söyler.

- **Bütün hataları körlemesine yakalamak.** except: pass gibi türü belirtilmemiş ve içi boş bir blok, bütün hataları sessizce yutar. Program hata vermez ama yanlış çalışır ve neden yanlış çalıştığını kimse bilemez. Her zaman beklediğin hata türünü yaz ve yakaladığın hatayla bir şey yap: kullanıcıya mesaj ver ya da kayda geçir.
- **try bloğunu gereğinden geniş tutmak.** Yirmi satırı tek bir try içine koyarsan hatanın hangisinden geldiğini anlamak zorlaşır. try bloğunda yalnız hata çıkarabilecek satırları tut.
- **Kullanıcıya teknik hata mesajı göstermek.** Traceback'i son kullanıcıya göstermek hem kafa karıştırır hem de dosya yolları gibi iç bilgileri dışarıya açabilir. Kullanıcıya anlaşılır bir mesaj ver, teknik ayrıntıyı log'a yaz.

Kendini dene

1. Sözdizimi hatası ile mantık hatası arasındaki fark nedir?
2. Bir traceback'i okumaya nereden başlarsın?
3. int("on iki") ifadesi hangi hata türünü üretir?
4. try-except yapısında finally bloğu ne zaman çalışır?
5. Türü belirtilmemiş ve içi boş bir except bloğu neden sakıncalıdır?

Web nasıl çalışır?

Bugün yazılan uygulamaların büyük kısmı bir şekilde web ile ilişkilidir: tarayıcıda açılan siteler, telefon uygulamalarının konuştuğu sunucular, firmaların birbirine veri gönderdiği servisler.

İstemci ve sunucu

Web, istemci-sunucu modeliyle çalışır. **İstemci (client)** bir istekte bulunan taraftır: tarayıcın, telefonundaki uygulama ya da başka bir program. **Sunucu (server)** isteği karşılayan taraftır: bir veri merkezinde ya da bulut ortamında sürekli çalışan, istekleri bekleyen bir bilgisayar ve üzerindeki yazılım. İstemci istek gönderir, sunucu yanıt verir.

Bir web adresinin (URL) parçalarına bakalım: <https://www.example.com/urunler?kategori=termos>

- **https:** Protokol. Tarayıcı ile sunucunun hangi kurallarla konuşacağını belirtir.
- **www.example.com:** Alan adı (domain). Sunucunun insanlar için hatırlanabilir adı.
- **/urunler:** Yol (path). Sunucudaki hangi sayfa ya da kaynağın istendiği.
- **?kategori=termos:** Sorgu parametresi (query string). İsteğe eklenen ek bilgi; burada yalnız termos kategorisinin istendiğini söylüyor.

Bir adrese girdiğinde ne olur?

1. **DNS sorgusu:** Bilgisayarlar birbirini alan adıyla değil, IP adresiyle bulur. Tarayıcı, alan adının hangi IP adresine karşılık geldiğini DNS (Domain Name System) sisteminden öğrenir. DNS'i internetin telefon rehberi gibi düşünebilirsiniz.
2. **Bağlantı:** Tarayıcı o IP adresindeki sunucuyla bağlantı kurar. Adres https ile başlıyorsa bu aşamada TLS ile şifreli bir kanal oluşturulur.
3. **HTTP isteği:** Tarayıcı, istediği sayfayı belirten bir HTTP isteği gönderir.
4. **Sunucunun işi:** Sunucudaki uygulama isteği işler. Gerekirse veri tabanını sorgular ve sayfayı hazırlar.
5. **HTTP yanıtı:** Sunucu bir durum kodu ve içerikle (çoğu zaman bir HTML belgesi) yanıt verir.
6. **Sayfanın çizilmesi:** Tarayıcı HTML'i okur, içinde adı geçen CSS, JavaScript ve görsel dosyalarını ayrı istekler halinde ister ve sayfayı ekrana çizer.

HTTP: web'in dili

HTTP (HyperText Transfer Protocol), istemci ile sunucu arasındaki iletişimin kurallarıdır. Her HTTP isteğinin bir yöntemi (method) vardır. Yöntem, istemcinin ne yapmak istediğini söyler:

- **GET:** Veri iste. Bir sayfayı açmak, ürün listesini almak.
- **POST:** Yeni veri gönder. Form göndermek, yeni sipariş oluşturmak.
- **PUT ve PATCH:** Var olan veriyi güncelle. PUT genellikle kaydın tamamını, PATCH bir kısmını değiştirir.
- **DELETE:** Veriyi sil.

Sunucunun yanıtı üç haneli bir durum koduyla başlar. İlk hane kodun sınıfını gösterir:

- **2xx, başarılı:** 200 OK (istek başarılı), 201 Created (yeni kayıt oluşturuldu).
- **3xx, yönlendirme:** 301 Moved Permanently (sayfa kalıcı olarak başka adrese taşındı).
- **4xx, istemci hatası:** 400 Bad Request (istek hatalı biçimde), 401 Unauthorized (kimlik doğrulama gerekli), 403 Forbidden (yetkin yok), 404 Not Found (istenen kaynak yok).

- **5xx, sunucu hatası:** 500 Internal Server Error (sunucuda beklenmeyen bir hata oluştu), 503 Service Unavailable (sunucu şu an hizmet veremiyor).

Hata ararken bu ayrım işe yarar: 4xx sorunun istekte (adres, veri ya da yetki), 5xx sunucu tarafında olduğunu söyler.

HTTP durumsuz (stateless) bir protokoldür: sunucu her isteği ayrı değerlendirir. Giriş yaptıktan sonra tekrar şifre sorulmamasını, tarayıcının her istekle gönderdiği çerezler (cookie) sağlar.

HTTPS, HTTP'nin TLS ile şifrelenmiş halidir. Tarayıcı ile sunucu arasındaki veriyi aradaki kişilerin okumasını ve değiştirmesini engeller. Şifre, kart bilgisi ya da kişisel veri içeren hiçbir form şifresiz HTTP üzerinden gönderilmemelidir.

HTML, CSS ve JavaScript

Tarayıcının anladığı üç temel dil vardır ve her birinin ayrı bir görevi bulunur. Bir binaya benzetirsek: HTML binanın iskeleti ve odalarıdır, CSS boyası ve döşemesidir, JavaScript ise elektrik tesisatı, yani düğmeye basınca bir şey olmasını sağlayan kısımdır.

HTML (HyperText Markup Language) sayfanın içeriğini ve yapısını etiketlerle tanımlar: başlık, paragraf, liste, bağlantı, görsel, form, düğme.

```
<!DOCTYPE html>
<html lang="tr">
<head>
  <meta charset="utf-8">
  <title>Randevu</title>
  <link rel="stylesheet" href="stil.css">
</head>
<body>
  <h1>Online randevu</h1>
  <p>Belediye hizmet binası, 2. kat</p>
  <button id="al">Randevu al</button>
  <p id="mesaj"></p> <!-- JavaScript mesajı buraya yazar -->
  <script src="uygulama.js"></script>
</body>
</html>
```

Etiketler genellikle açılış ve kapanış çiftiyle yazılır: h1 etiketi ana başlığı, p etiketi paragrafı, button etiketi düğmeyi tanımlar. head bölümündeki meta charset="utf-8" satırı, Türkçe karakterlerin doğru görünmesi için gereklidir. id özelliği bir öğeye benzersiz bir ad verir; CSS ve JavaScript o öğeyi bu adla bulur.

CSS (Cascading Style Sheets) sayfanın görünüşünü belirler: renk, yazı tipi, boşluklar, yerleşim. Aynı HTML, farklı CSS ile bambaşka görünebilir.

```
/* stil.css: sayfanın görünümü */
body {
  font-family: Arial, sans-serif;
  margin: 24px;
}
h1 {
  color: #1f4e79; /* koyu mavi başlık */
}
#al {
```

```
padding: 8px 16px;
cursor: pointer;
}
```

Her CSS kuralı bir seçici (body, h1, #al) ve süslü parantez içinde özellik-değer çiftlerinden oluşur. Diyez işaretli seçici, id'si "al" olan öğeyi seçer. CSS ile sayfanın telefonda ve bilgisayarda farklı yerleşmesini de sağlarsın; buna duyarlı tasarım (responsive design) denir.

JavaScript sayfaya davranış ekler: düğmeye tıklanınca bir şey yapmak, formu göndermeden önce kontrol etmek, sayfayı yenilemeden sunucudan yeni veri almak.

```
// uygulama.js: düğmeye tıklanınca mesaj gösterir
const dugme = document.getElementById("al");
const mesaj = document.getElementById("mesaj");

dugme.addEventListener("click", function () {
  const saat = new Date().getHours();
  if (saat < 17) {
    mesaj.textContent = "Bugün için randevu alabilirsin.";
  } else {
    mesaj.textContent = "Mesai bitti, yarın için randevu alabilirsin.";
  }
});
```

JavaScript'te kod blokları Python'daki gibi girintiyle değil süslü parantezlerle belirlenir; koşullar parantez içine yazılır; satırlar genellikle noktalı virgülle biter. const sabit bir değişken, let değiştirilebilir bir değişken tanımlar. Mantık ise Python'da öğrendiklerinle aynıdır: değişken, koşul, fonksiyon.

Erişilebilirlik de front-end işinin parçasıdır: görsellere açıklayıcı metin, yeterli renk kontrastı ve klavyeyle kullanılabilen bir sayfa.

Back-end'in rolü

Kullanıcı formu doldurup "Randevu al" düğmesine bastığında tarayıcı sunucuya bir POST isteği gönderir. Sunucudaki back-end kodu veriyi ve yetkiyi kontrol eder, saatin boş olup olmadığını veri tabanından sorgular, randevuyu kaydeder ve sonucu bildirir. Python'da bu iş için Django, Flask ve FastAPI gibi çatılar kullanılır.

Önemli bir kural: istemciden gelen hiçbir veriye güvenilmez. Tarayıcıdaki JavaScript kontrolleri kullanıcıya hızlı geri bildirim vermek için iyidir, ama kötü niyetli biri bu kontrolleri kolayca atlayıp sunucuya doğrudan istek gönderebilir. Fiyat, yetki, stok gibi bütün kritik kontroller sunucuda tekrar yapılmalıdır.

API

API (Application Programming Interface, uygulama programlama arayüzü), programların birbiriyle hangi kurallarla konuşacağını tanımlar.

Web API'leri, HTTP üzerinden çalışır. Bir e-ticaret sitesi kargo firmasının API'sine gönderi oluşturma isteği gönderir; bir mobil uygulama kendi şirketinin API'sinden ürün listesini alır. Yaygın bir tasarım tarzı olan REST'te her kaynak bir adresle, yapılacak işlem HTTP yöntemiyle ifade edilir:

- GET /api/siparisler: Siparişleri listele.
- GET /api/siparisler/1042: 1042 numaralı siparişi getir.

- POST /api/siparisler: Yeni sipariş oluştur.
- PATCH /api/siparisler/1042: 1042 numaralı siparişin bir alanını güncelle.
- DELETE /api/siparisler/1042: 1042 numaralı siparişi sil.

Birçok API, kimin istek gönderdiğini anlamak için bir API anahtarı (API key) ya da erişim belirteci (token) ister. Bu anahtarlar şifre gibidir: tarayıcıda çalışan JavaScript koduna yazılmaz, kod deposuna gönderilmez, e-postayla paylaşılmaz.

JSON

API'ler veriyi çoğunlukla JSON (JavaScript Object Notation) biçiminde gönderir. JSON, insanların da okuyabildiği düz metin bir veri biçimidir ve Python'daki sözlük ile listelere çok benzer.

```
# API'den gelmiş gibi bir JSON yanıtını işlemek
import json

yanit = '{"siparis_no": 1042, "durum": "kargoda", "teslim_edildi": false, ' \
        ' "urunler": [{"ad": "Termos", "adet": 2}, {"ad": "Kupa", "adet": 1}]}'
veri = json.loads(yanit)          # JSON metni -> Python sözlüğü
print(veri["durum"], veri["teslim_edildi"]) # kargoda False
for u in veri["urunler"]:
    print(u["ad"], u["adet"])

yeni = {"siparis_no": 1043, "durum": "hazırlanıyor"}
print(json.dumps(yeni, ensure_ascii=False)) # Python sözlüğü -> JSON metni
```

json.loads() JSON metnini Python yapısına, json.dumps() Python yapısını JSON metnine çevirir. ensure_ascii=False, Türkçe karakterlerin kodlanmadan olduğu gibi yazılmasını sağlar. JSON ile Python arasındaki farklara dikkat et: JSON'da metinler yalnız çift tırnakla yazılır, mantıksal değerler küçük harfle true ve false'tur, boş değer null'dır. Python'a geçerken bunlar True, False ve None olur. JSON'da son elemandan sonra virgöl konamaz.

Python'da bir API'ye gerçek istek göndermek için genellikle requests paketi kullanılır; requests.get(adres) ile istek gönderilir, gelen yanıtın json() metoduyla içerik sözlüğe çevrilir. Tarayıcıdaki JavaScript ise aynı işi fetch() fonksiyonuyla yapar.

Uygulama

Tarayıcının geliştirici araçları, web'in nasıl çalıştığını kendi gözünle görmenin en kolay yoludur. Chrome, Edge ve Firefox'ta F12 tuşuyla (macOS'ta Cmd+Option+I) açılır.

1. Sık kullandığın bir haber ya da e-ticaret sitesini aç ve geliştirici araçlarını aç.
2. **Elements (Öğeler)** sekmesinde sayfanın HTML yapısını ve uygulanan CSS kurallarını gör. Bir rengi değiştirmeyi dene; değişiklik yalnız senin ekranında ve sayfayı yenileyene kadar geçerlidir.
3. **Network (Ağ)** sekmesini aç ve sayfayı yenile. Gönderilen istekleri göreceksin; birine tıklayıp yöntemini (GET) ve durum kodunu (200) incele. Türev fetch ya da XHR olan isteklerin çoğu API çağrısıdır; yanıtlarında JSON verisi görebilirsin.
4. **Console (Konsol)** sekmesinde document.title yazıp Enter'a bas; sayfanın başlığını görürsün.

Sık yapılan hatalar

- **Doğrulamayı yalnız front-end'de yapmak.** Tarayıcıdaki kontroller atlanabilir; kritik kontroller sunucuda tekrarlanmalıdır.
- **API anahtarını istemci koduna ya da kod deposuna koymak.** Tarayıcıya gönderilen her şey kullanıcılar tarafından görülebilir.
- **HTML'de karakter kodlamasını belirtmemek.** meta charset="utf-8" satırı olmayan sayfalarda Türkçe karakterler bozuk görünebilir.
- **Kişisel veriyi adres çubuğundaki parametrelere koymak.** Sorgu parametreleri tarayıcı geçmişinde ve sunucu kayıtlarında görünür. Bu tür veriler isteğin gövdesinde, HTTPS üzerinden gönderilir.

Kendini dene

1. Tarayıcıya bir adres yazdığında alan adını IP adresine çeviren sistem hangisidir?
2. 404 ve 500 durum kodları neyi anlatır, hangisi sunucu tarafındaki bir hatayı gösterir?
3. HTML, CSS ve JavaScript'in görevlerini birer cümleyle açıkla.
4. Yeni bir kayıt oluşturmak için genellikle hangi HTTP yöntemi kullanılır?
5. JSON'daki true ve null değerleri Python'a hangi değerler olarak geçer?

Veri tabanına giriş ve SQL'in temeli

Neredeyse her iş uygulamasının arkasında bir veri tabanı vardır: e-ticaret sitesinin ürünleri ve siparişleri, hastanenin randevuları, belediyenin abone kayıtları, şirketin personel bilgileri. Bu modülde verinin neden dosyada değil veri tabanında tutulduğunu ve SQL'in temellerini öğreneceksin.

Neden veri tabanı?

Küçük bir işletme müşterilerini bir Excel dosyasında tutabilir. Ama iş büyüdükçe dosyanın sınırları ortaya çıkar:

- **Aynı anda kullanım:** On kişi aynı dosyayı aynı anda güncellemeye çalışırsa değişiklikler birbirinin üzerine yazılır ya da dosya kilitlenir.
- **Tutarlılık:** Bir müşterinin adresi üç farklı sayfada tutuluyorsa biri güncellenir, diğerleri eski kalır.
- **Yetki:** Satış ekibinin müşteri adlarını görmesi ama kimlik bilgilerini görmemesi gerekebilir. Dosyada bu ayrımı yapmak zordur.

Veri tabanı, verilerin düzenli, tutarlı ve güvenli biçimde saklandığı yapıdır. Veri tabanını yöneten yazılıma veri tabanı yönetim sistemi (DBMS) denir. Yaygın ilişkisel veri tabanı yönetim sistemleri şunlardır: PostgreSQL, MySQL, Microsoft SQL Server, Oracle Database ve SQLite. SQLite, ayrı bir sunucu gerektirmeyen, tek bir dosyada çalışan hafif bir sistemdir ve Python ile birlikte gelir; bu yüzden öğrenmek için çok uygundur.

İlişkisel veri tabanı: tablolar ve anahtarlar

İlişkisel veri tabanında veri tablolar halinde tutulur. Her tablo bir konuyu anlatır: müşteriler, ürünler, siparişler.

- **Satır (kayıt):** Tablodaki tek bir öge. Bir müşteri, bir sipariş.
- **Sütun (alan):** Her kaydın sahip olduğu bir bilgi. Müşteri adı, şehir, kayıt tarihi. Her sütunun bir veri tipi vardır: tam sayı, metin, tarih, ondalıklı sayı.
- **Birincil anahtar (primary key):** Her kaydı benzersiz biçimde tanımlayan sütun. Genellikle bir numara: müşteri numarası, sipariş numarası. İki kaydın birincil anahtarı aynı olamaz.
- **Yabancı anahtar (foreign key):** Bir tablodaki sütunun başka bir tablonun birincil anahtarını göstermesi. Siparişler tablosundaki müşteri_id sütunu, siparişi hangi müşterinin verdiğini müşteriler tablosuna bakarak söyler.

Bu yapı sayesinde müşterinin adı ve şehri yalnız bir yerde, müşteriler tablosunda tutulur. Siparişlerde yalnız müşteri numarası bulunur. Müşteri şehrini değiştirdiğinde tek bir kaydı güncelleme yeterlidir. Verinin tekrarlanmadan, anlamlı tablolara bölünmesine normalizasyon denir.

İlişkisel olmayan (NoSQL) veri tabanları da vardır; belge tabanlı MongoDB ve anahtar-değer tabanlı Redis bunlara örnektir. Başlangıç için ilişkisel veri tabanlarını ve SQL'i öğrenmek en sağlam temeldir.

SQL nedir?

SQL (Structured Query Language), ilişkisel veri tabanlarıyla konuşmak için kullanılan standart dildir. Veri tabanından veri okumak, eklemek, güncellemek, silmek ve tablo oluşturmak için kullanılır. Temel komutlar bütün sistemlerde aynıdır; bazı ayrıntılar (tarih fonksiyonları, sonuç sayısını sınırlama yöntemi gibi) sistemden sisteme değişir. Anahtar kelimeleri büyük harfle yazmak yaygın bir alışkanlıktır.

Tablo oluşturmak ve veri eklemek

```
-- Müşteriler ve siparişler tabloları
CREATE TABLE musteriler (
    id INTEGER PRIMARY KEY,
    ad TEXT NOT NULL,
    sehir TEXT
);
CREATE TABLE siparisler (
    id INTEGER PRIMARY KEY,
    musteri_id INTEGER REFERENCES musteriler(id),
    tarih DATE,
    tutar DECIMAL(10, 2)
);
```

NOT NULL, o sütunun boş bırakılamayacağını söyler. REFERENCES, musteri_id sütununun müşteriler tablosundaki id'yi gösteren bir yabancı anahtar olduğunu belirtir. DECIMAL(10, 2) toplam 10 basamaklı, 2'si ondalık olan bir sayı demektir; para tutarları için uygundur.

```
-- Kayıt eklemek
INSERT INTO musteriler (id, ad, sehir) VALUES (1, 'Deniz', 'Ankara');
INSERT INTO musteriler (id, ad, sehir) VALUES (2, 'Burak', 'İzmir');
INSERT INTO musteriler (id, ad, sehir) VALUES (3, 'Zeynep', 'Ankara');
INSERT INTO siparisler (id, musteri_id, tarih, tutar)
VALUES (101, 1, '2026-10-02', 1450.00),
       (102, 2, '2026-10-05', 320.50),
       (103, 1, '2026-10-11', 2780.00);
```

SQL'de metin ve tarih değerleri tek tırnakla yazılır. Her komut noktalı virgülle biter. Satır başındaki iki tire (--) yorum başlatır.

Veri okumak: SELECT

SQL'de en çok kullanılan komut SELECT'tir. Hangi sütunların, hangi tablodan, hangi koşulla ve hangi sırayla getirileceğini söylersin.

```
-- Ankara'daki müşterileri ada göre sıralı getirir
SELECT id, ad
FROM musteriler
WHERE sehir = 'Ankara'
ORDER BY ad;
```

- **SELECT:** Getirilecek sütunlar. Bütün sütunlar için yıldız işareti kullanılır, ama gerçek uygulamalarda yalnız ihtiyacın olan sütunları yazmak daha iyidir.
- **FROM:** Verinin alınacağı tablo.
- **WHERE:** Koşul. Yalnız koşulu sağlayan satırlar gelir. Karşılaştırmalar için =, <> (eşit değil), >, <, >=, <= kullanılır; koşullar AND, OR ve NOT ile birleştirilir.
- **ORDER BY:** Sıralama. Varsayılan küçükten büyüğe (ASC); büyükten küçüğe için DESC yazılır.

WHERE içinde kullanılan bazı yararlı ifadeler:

- **BETWEEN:** Aralık. tutar BETWEEN 500 AND 2000.
- **IN:** Listeden biri. sehir IN ('Ankara', 'İzmir').
- **LIKE:** Metin kalıbı. ad LIKE 'Z%' adı Z ile başlayanları getirir; yüzde işareti "herhangi bir karakter dizisi" anlamına gelir.
- **IS NULL:** Boş değer kontrolü. sehir IS NULL. Boş değerler eşittir işaretiyle kontrol edilemez.

Özetlemek: toplama fonksiyonları ve GROUP BY

SQL veriyi yalnız listelemez, özetler de. COUNT (sayı), SUM (toplam), AVG (ortalama), MIN ve MAX (en küçük ve en büyük) toplama fonksiyonlarıdır. GROUP BY ile veriyi gruplara ayırıp her grup için ayrı özet alırsın.

```
-- Şehir başına müşteri sayısı
SELECT sehir, COUNT(*) AS musteri_sayisi
FROM musteriler
GROUP BY sehir
ORDER BY musteri_sayisi DESC;
```

AS, sonuç sütununa okunaklı bir ad verir. Gruplanmış sonuçlar üzerinde koşul koymak için WHERE değil HAVING kullanılır: "toplam siparişi 5.000 TL'yi geçen müşteriler" gibi.

Tabloları birleştirmek: JOIN

Veri tablolara bölündüğü için çoğu soruyu cevaplamak üzere tabloları birleştirmen gerekir. "Siparişleri müşteri adlarıyla birlikte göster" sorusunun cevabı iki tablodadır.

```
-- 1.000 TL ve üzeri siparişleri müşteri adlarıyla getirir
SELECT m.ad, s.tarih, s.tutar
FROM siparisler AS s
JOIN musteriler AS m ON m.id = s.musteri_id
WHERE s.tutar >= 1000
ORDER BY s.tarih;
```

JOIN ... ON, iki tablonun hangi sütunlar üzerinden eşleşeceğini söyler. Tablolara verilen kısa adlar (s ve m), sorguyu kısaltır. Bu şekilde yazılan JOIN (INNER JOIN), yalnız iki tabloda da eş bulunan satırları getirir. LEFT JOIN ise soldaki tablonun bütün satırlarını getirir, sağda eş olmayanların sütunlarını boş bırakır.

Güncellemek ve silmek

```
-- Bir müşterinin şehrini günceller
UPDATE musteriler SET sehir = 'Eskişehir' WHERE id = 2;

-- Tek bir siparişi siler
DELETE FROM siparisler WHERE id = 102;
```

UPDATE ve DELETE komutlarında WHERE koşulu hayati önem taşır. WHERE yazmayı unutursan UPDATE bütün müşterilerin şehrini Eskişehir yapar, DELETE bütün siparişleri siler. Bu tür komutları çalıştırmadan önce aynı WHERE koşuluyla bir SELECT çalıştırıp hangi satırların etkileneceğini görmek iyi bir alışkanlıktır.

Python ile veri tabanı

Python'un standart kütüphanesindeki sqlite3 modülü, ek kurulum gerektirmeden SQLite veri tabanıyla çalışmanı sağlar.

```
# Python'un yerleşik sqlite3 modülüyle küçük bir veri tabanı
import sqlite3

baglanti = sqlite3.connect(":memory:") # bellekte geçici veri tabanı
imlec = baglanti.cursor()
imlec.execute("CREATE TABLE urunler (kod TEXT PRIMARY KEY, ad TEXT, stok INTEGER)")
veriler = [("K1", "Kalem", 120), ("D1", "Defter", 45), ("S1", "Silgi", 8)]
imlec.executemany("INSERT INTO urunler VALUES (?, ?, ?)", veriler)
baglanti.commit()

esik = 50
sorgu = "SELECT kod, ad, stok FROM urunler WHERE stok < ? ORDER BY stok"
for kod, ad, stok in imlec.execute(sorgu, (esik,)):
    print(kod, ad, stok)
baglanti.close()
```

":memory:" veri tabanını bellekte oluşturur ve program bitince siler; denemeler için uygundur. Kalıcı bir veri tabanı için yerine "depo.db" gibi bir dosya adı yazarsın. commit(), yapılan değişiklikleri kalıcı hale getirir. Sorgudaki soru işaretleri yer tutuculardır; gerçek değerler ayrı bir argüman olarak verilir.

Bu son nokta güvenlik açısından çok önemlidir. Kullanıcıdan gelen bir değeri sorgu metnine doğrudan eklersen, örneğin bir f-string ile "WHERE ad = '{ad}'" biçiminde yazarsan, kötü niyetli bir kullanıcı ad yerine SQL komutu yazarak sorguyu değiştirebilir ve yetkisi olmayan verilere ulaşabilir ya da veri silebilir. Buna SQL enjeksiyonu (SQL injection) denir ve web uygulamalarındaki en bilinen güvenlik açıklarından biridir. Çözüm her zaman aynıdır: kullanıcı girdisini sorguya yer tutucularla, yani parametrelili sorguyla ver.

Uygulama

Bursa'da bir kırtasiye toptancısının satış müdürü iki soru soruyor. Her soruyu önce düz Türkçeyle düşün, sonra SQL'e çevir.

Soru 1: En çok harcama yapan beş müşteri kim? Siparişleri müşteriye göre gruplayıp tutarları toplarsın, toplama göre sıralar ve ilk beşi alırsın.

```
-- En çok harcama yapan 5 müşteri
SELECT m.ad, SUM(s.tutar) AS toplam
FROM musteriler AS m
JOIN siparisler AS s ON s.musteri_id = m.id
GROUP BY m.id, m.ad
ORDER BY toplam DESC
LIMIT 5;
```

LIMIT, SQLite, PostgreSQL ve MySQL'de sonuç sayısını sınırlar. Microsoft SQL Server'da aynı iş SELECT TOP 5 biçiminde yazılır.

Soru 2: Hiç sipariş vermemiş müşteriler kimler? Burada LEFT JOIN kullanırsın: bütün müşterileri getirir, siparişi olmayanların sipariş sütunları boş kalır. Sonra boş olanları seçersin.

```
-- Hiç sipariş vermemiş müşteriler
SELECT m.ad
FROM musteriler AS m
LEFT JOIN siparisler AS s ON s.musteri_id = m.id
WHERE s.id IS NULL;
```

Silme örneğini çalıştırmadıysan, yukarıdaki örnek verilerle bu sorgu Zeynep'i getirir. SQL'i öğrenmenin en iyi yolu, gerçek iş sorularını sorguya çevirmektir. Tabloları SQLite'ta oluştur ve kendine yeni sorular sor: "Her şehrin toplam sipariş tutarı ne?", "Ekim ayındaki siparişler hangileri?"

Veri tabanında kişisel veri tutuyorsan 6698 sayılı KVKK kapsamındaki yükümlülükler de işin parçasıdır: yalnız gerekli veriyi tut, erişimi yetkiye göre sınırla, düzenli yedek al ve gereğinden uzun saklama. Deneme ortamlarında gerçek müşteri verisi kullanma.

Sık yapılan hatalar

- **UPDATE ve DELETE'te WHERE'i unutmak.** Bütün tablo etkilenir. Önce aynı koşulla SELECT çalıştır, canlı veri tabanında çalışmadan önce yedek olduğundan emin ol.
- **Kullanıcı girdisini sorgu metnine eklemek.** SQL enjeksiyonuna yol açar. Her zaman parametrelili sorgu kullan.
- **Boş değeri eşittir ile aramak.** WHERE şehir = NULL hiçbir satır getirmez; doğrusu WHERE şehir IS NULL.
- **Metinlerde çift tırnak kullanmak.** Standart SQL'de metin değerleri tek tırnakla yazılır; çift tırnak birçok sistemde sütun ya da tablo adı anlamına gelir.

Kendini dene

1. Birincil anahtar (primary key) nedir?
2. Ankara'daki müşterilerin adlarını getiren SQL sorgusunu yaz.
3. WHERE içermeyen bir DELETE FROM siparisler komutu ne yapar?
4. Bir sütunun boş (NULL) olup olmadığını hangi ifadeyle kontrol edersin?
5. SQL enjeksiyonunu önlemek için Python'da sorgulara kullanıcı girdisi nasıl verilmelidir?

Git, takım çalışması, test ve kariyer yolları

Tek başına küçük betikler yazarken kodunu bir klasörde tutman yeterlidir. Ekip halinde çalışırken ya da kodun eski bir haline dönmek gerektiğinde daha düzenli bir yöntem ihtiyacı duyarsın. Bu modül sürüm kontrolünü, ekip çalışmasını, testi ve kariyer yollarını anlatır.

Sürüm kontrolü ve Git

"rapor_son.docx", "rapor_son_SON_v3.docx" gibi dosya adlarını tanıyorsan sürüm kontrolüne neden ihtiyaç duyulduğunu biliyorsun. Sürüm kontrol sistemi, dosyalardaki her değişikliği kimin, ne zaman ve neden yaptığıyla birlikte kaydeder; istediğin zaman eski bir hale dönebilirsin.

Bugün en yaygın sürüm kontrol sistemi Git'tir. 2005'te Linus Torvalds tarafından Linux çekirdeğinin geliştirilmesi için yazılmıştır. Git dağıtık bir sistemdir: her geliştiricinin bilgisayarında projenin bütün geçmişini içeren bir kopya bulunur. GitHub, GitLab ve Bitbucket ise Git depolarını internette barındıran ve ekip çalışması için ek araçlar sunan hizmetlerdir. Git bir araçtır, GitHub bu aracın kullanıldığı bir hizmettir.

Temel kavramlar:

- **Depo (repository):** Projenin dosyalarını ve bütün değişiklik geçmişini içeren yapı.
- **Commit:** Belirli bir andaki değişikliklerin, açıklayıcı bir mesajla kaydedilmiş hali.
- **Hazırlık alanı (staging area):** Bir sonraki commit'e girecek değişikliklerin seçildiği ara alan.
- **Dal (branch):** Ana koddan ayrılıp bağımsız geliştirilen çalışma hattı. Ana dal genellikle main adını taşır.
- **Birleştirme (merge):** Bir daldaki değişiklikleri başka bir dala aktarmak.
- **Uzak depo (remote), clone, push, pull:** Deponun GitHub gibi bir sunucudaki ortak kopyası; onu bilgisayarına kopyalamak (clone), kendi commit'lerini göndermek (push), başkalarının değişikliklerini almak (pull).

İlk Git adımları

Git'i kurduktan sonra ilk iş, commit'lerde görünecek adını ve e-posta adresini git config --global user.name ve git config --global user.email komutlarıyla tanımlamaktır. Sonra bir proje klasöründe:

```
# Yeni bir depo başlatıp ilk kaydı oluşturmak
git init                    # klasörü Git deposu yapar
git status                  # değişen ve izlenmeyen dosyaları gösterir
git add hesaplar.py        # dosyayı hazırlık alanına ekler
git commit -m "KDV ve indirim hesaplama fonksiyonları eklendi"
git log --oneline          # kayıt geçmişini kısa gösterir
```

Yeni bir özellik üzerinde ayrı bir dalda çalışırsın:

```
# Yeni özellik için dal açıp uzak depoya göndermek
git switch -c kargo-hesabi  # yeni dal oluşturup ona geçer
# ... dosyaları düzenle ...
git add kargo.py
git commit -m "Desiye göre kargo ücreti hesabı eklendi"
git push -u origin kargo-hesabi # dalı uzak depoya gönderir
```

Eski kaynaklarda git switch -c yerine git checkout -b biçimini görebilirsin. İyi bir commit mesajı ne yapıldığını kısa ve açık söyler: "Kargo ücretinde şehir içi indirimi eklendi" iyi bir mesajdır; "düzeltme" ya da "son hali" değildir. Commit'leri küçük ve tek konulu tut.

Şifre ve API anahtarı içeren ayar dosyaları (.env gibi), sanal ortam klasörü ve geçici dosyalar depoya girmemelidir. Bunların adları .gitignore dosyasına yazılır. Bir şifreyi yanlışlıkla commit'lersen dosyayı sonradan silmek yetmez; şifre geçmişteki commit'te durur. Şifre hemen değiştirilmeli, yani eskisi geçersiz kılınmalıdır.

Takım çalışması

Ekipler genellikle şöyle çalışır: her iş bir görev kaydı (issue) olarak açılır. Geliştirici o iş için bir dal açar, kodunu yazar ve uzak depoya gönderir. Sonra bir **pull request** (GitLab'de merge request) açar: "Bu dalı ana dala birleştirmek istiyorum, lütfen inceleyin." Ekipten biri kodu inceler (code review), soru sorar, düzeltme önerir. İnceleme onaylanınca ve otomatik testler geçince dal ana dala birleştirilir. Kod incelemesi bilgiyi ekipte yayar ve yeni başlayanlar için etkili bir öğrenme yoludur. İnceleme yaparken kişiyi değil kodu konuş: "Bu fonksiyon boş liste gelirse sıfıra bölme hatası verebilir" gibi.

İki kişi aynı dosyanın aynı satırlarını farklı biçimde değiştirirse Git hangisinin doğru olduğuna karar veremez. Buna çakışma (merge conflict) denir ve Git dosyada iki sürümü işaretler:

```
<<<<<< HEAD
SABIT_KARGO = 70
=====
SABIT_KARGO = 75
>>>>>> yeni-tarife
```

Üstteki kısım senin dalındaki, alttaki birleştirilen daldaki halidir. Doğru olanı (gerekirse ilgili kişiyle konuşarak) seçer, işaret satırlarını siler ve commit'lersin.

Ekipler ayrıca ortak bir kodlama standardında anlaşır; Python için yaygın stil rehberi PEP 8'dir. Hata bildirirken hangi adımlarla oluştuğu, ne beklendiği, ne görüldüğü ve hangi ortamda (tarayıcı, işletim sistemi, uygulama sürümü) olduğu yazılır. "Çalışmıyor" bir hata bildirimi değildir.

Test

Test, yazılımın beklendiği gibi çalıştığını kontrol etmektir. Elle testte bir kişi senaryoları dener; otomatik testte kodu başka bir kod kontrol eder ve her değişiklikte saniyeler içinde yeniden çalıştırılır. Testler kapsamına göre ayrılır:

- **Birim testi (unit test):** Tek bir fonksiyonu ya da küçük bir kod parçasını ayrı olarak test eder. Hızlıdır, sayıca en çok olan testlerdir.
- **Entegrasyon testi:** Birden fazla parçanın, örneğin kodun veri tabanı ile, birlikte doğru çalıştığını test eder.
- **Uçtan uca (end-to-end) test:** Uygulamayı gerçek bir kullanıcı gibi baştan sona dener: giriş yap, sepete ekle, ödemeyi tamamla.

Bir değişikliğin daha önce çalışan bir şeyi bozup bozmadığını kontrol etmeye regresyon testi denir. Python'un standart kütüphanesindeki unittest modülüyle Modül 5'teki kargo fonksiyonunun basitleştirilmiş bir sürümünü test edelim:

```
# Kargo ücreti fonksiyonu için birim testleri (unittest)
import unittest
```

```
def kargo_ucreti(desi):
    if desi <= 0:
        return None
    return 70 if desi <= 3 else 70 + (desi - 3) * 12
class KargoTestleri(unittest.TestCase):
    def test_kucuk_paket(self):
        self.assertEqual(kargo_ucreti(2), 70)
    def test_sinir_degeri(self):
        self.assertEqual(kargo_ucreti(3), 70)
    def test_gecersiz_desi(self):
        self.assertIsNone(kargo_ucreti(0))
unittest.main()
```

Her test metodu bir durumu dener. Hepsi geçerse "OK" yazılır; biri başarısız olursa hangi testin hangi değerle başarısız olduğu raporlanır. Modül 2'de fazla mesai için hazırladığın deneme listesi aslında bir test listesiydi. pytest paketi de yaygındır.

Ekiplerde testler, kod uzak depoya her gönderildiğinde otomatik olarak çalıştırılır. Buna sürekli entegrasyon (CI, Continuous Integration) denir; testleri geçmeyen kod ana dala birleştirilmez.

Örnek vaka

Adana'da bir hastanenin bilgi işlem biriminde yeni işe başlayan bir yardımcı geliştiriciye ilk görev veriliyor: randevu hatırlatma betiği yalnız bir gün önce mesaj gönderiyor; poliklinik sorumluları randevudan iki saat önce de hatırlatma istiyor.

1. Geliştirici görevde belirsiz bir nokta görüyor: sabah erken saatlerdeki randevular için iki saat önceki mesaj gece mi gidecek? Poliklinik sorumlusuna soruyor; cevap, gece mesaj gönderilmemesi.
2. Depoyu klonluyor ve hatırlatma-iki-saat adında bir dal açıyor.
3. Hatırlatma zamanını hesaplayan fonksiyonu yazıyor ve birim testlerini ekliyor: normal randevu, sabah 08.30 randevusu, iptal edilmiş randevu.
4. Pull request açıyor. Kıdemli geliştirici iki not düşüyor: hata ayıklama için eklenen log satırı hastanın TC kimlik numarasını yazıyor, bu da kişisel verinin log dosyasına kopyalanması demek; ayrıca saat dilimi hesabı sunucunun ayarına bağımlı.
5. Geliştirici log'da kimlik numarası yerine iç randevu numarasını kullanıyor, saat dilimini açıkça belirtiyor ve yeni commit'leri aynı dala gönderiyor.
6. Sürekli entegrasyon testleri geçiyor, inceleme onaylanıyor, dal ana dala birleştiriliyor. Değişiklik önce test ortamında deneme verileriyle çalıştırılıyor, sonra canlıya alınıyor.

Bu vakada kod yazmak işin yalnızca bir parçası; belirsizliği sormak, test yazmak ve kişisel verilere dikkat etmek de işin içinde.

Kariyer yolları ve no-code araçlar

Bu eğitimde temellerini gördüğün konular farklı uzmanlıklara çıkar:

- **Front-end:** HTML, CSS ve JavaScript'i derinleştir, ardından React, Vue ya da Angular'dan birini öğren.
- **Back-end:** Bir dil (Python, Java, C#, JavaScript, Go) ve onun bir web çatısı; SQL, API tasarımı ve güvenlik.
- **Test ve kalite güvencesi:** Elle testten test otomasyonuna.

- **Veri, DevOps ve bulut:** Python ve SQL ile veri analizi; Linux, ağ temelleri ve bulut platformları (AWS, Microsoft Azure, Google Cloud).

No-code araçlar kod yazmadan, sürükle-bırak arayüzlerle form, uygulama ve iş akışı oluşturmayı; low-code araçlar ise gerektiğinde kod eklemeyi sağlar. Microsoft Power Apps, Power Automate ve Zapier bu gruba girer. Avantajları: kurum içi küçük ihtiyaçlar (izin formu, onay akışı) hızla çözülür, yazılım ekibi beklenmeden prototip yapılır, iş birimleri kendi süreçlerini düzenleyebilir. Sınırları: karmaşık iş kurallarında zorlanır, kullanım büyüdükçe maliyet artabilir, uygulama platforma bağımlı kalır ve verinin nerede saklandığı KVKK açısından değerlendirilmelidir. Bu eğitimde öğrendiklerin burada da işe yarar: bir otomasyon aracında "tutar 10.000'den büyükse onaya gönder" adımını kurmak, bir if bloğu yazmaktır.

Yapay zekâ destekli kod asistanları da yararlıdır, ama üretilen kodun sorumluluğu onu kullandıdır: anlamadan kullanma, onaylanmamış araçlara gizli kod ya da kişisel veri verme.

Nasıl ilerlemeli?

1. **Bir dilde derinleş.** Aynı anda birçok dile başlamak hiçbirinde ilerleyememeye yol açar.
2. **Kendi problemini çöz.** İşinde tekrarlanan bir işi otomatikleştir: raporları birleştirmek, bir CSV'den özet çıkarmak.
3. **Yalnız izleyerek öğrenme.** Her konudan sonra kaynağa bakmadan kendin bir şey yaz.
4. **Projelerini GitHub'da tut.** Her birine ne yaptığını ve nasıl çalıştırıldığını anlatan bir README yaz; bu depolar portföyün olur.

Sık yapılan hatalar

- **Doğrudan ana dala kod göndermek.** Değişiklikler ayrı dalda yapılır ve incelemeden geçer.
- **Şifre ve API anahtarlarını depoya göndermek.** .gitignore kullan; yanlışlıkla gönderdiysen şifreyi hemen değiştir.
- **Anlamsız commit mesajları ve dev commit'ler.** "Düzeltilme" mesajlı, yüzlerce satırlık bir commit'in ne yaptığını kimse anlayamaz.
- **Testi sona bırakmak.** Fonksiyonu yazarken testini de yaz.
- **No-code'u ya hep ya hiç görmek.** Bazı işler için doğru çözüm no-code araçtır, bazıları için kod yazmak gerekir.

Kendini dene

1. git add ve git commit komutlarının görevleri nedir?
2. Yeni bir özellik için neden ayrı bir dal (branch) açılır?
3. Birim testi (unit test) neyi test eder?
4. Bir API anahtarı yanlışlıkla Git deposuna gönderildiyse dosyayı silmek neden yetmez, ne yapılmalıdır?
5. No-code araçların iki avantajını ve iki sınırını say.

Kendini dene cevapları

Modül 1: 1) Derlemede kodun tamamı çalıştırılmadan önce makine koduna çevrilir; yorumlamada kod çalışma sırasında yorumlayıcı tarafından okunup yürütülür. 2) İhtiyaç analizi, tasarım, geliştirme, test, yayına alma, bakım. 3) Kullanıcının ürünü çok geç görmesi; yanlış anlaşılmış bir ihtiyacın geç ortaya çıkması. 4) Front-end geliştirici. 5) Sprint.

Modül 2: 1) Açık, sıralı, sonlu, girdisi ve çıktısı belli, genel (herhangi üçü). 2) Algoritmayı bir dilin kurallarına bağlı kalmadan sade cümlelerle yazıp düşüncüyü netleştirmeye ve paylaşmaya. 3) Eşkenar dörtgen (baklava dilimi). 4) Sıra, seçim, tekrar. 5) İkili arama (binary search).

Modül 3: 1) Metin (str). 2) 3 ve 2. 3) adet_2. 4) {degisken:.2f}. 5) Python ondalık ayırıcı olarak noktayı tanır; çözüm float(metin.replace(".", "")).

Modül 4: 1) Girinti. 2) 2, 3, 4, 5. 3) for, dolaşılacak değerler ya da tekrar sayısı belli olduğunda; while, tekrar sayısı bilinmeyip bir koşul sağlandığı sürece çalışılacaksa. 4) continue. 5) Bronz; ilk doğru koşulun bloğu çalışır, diğerlerine bakılmaz.

Modül 5: 1) Parametre, fonksiyon tanımındaki addır; argüman, çağrıda verilen somut değerdir. 2) None. 3) Argüman verilmezse kullanılacak değeri belirler; kdv_dahil(1000) çağrısında oran 0.20 olur. 4) import random satırı Python'un modülü yerine senin dosyanı yükler. 5) pip.

Modül 6: 1) "Bursa". 2) KeyError oluşur; get() kullanmak ya da önce in ile kontrol etmek güvenlidir. 3) "w" içeriği silip baştan yazar; "a" içeriği koruyup sonuna ekler. 4) Blok bitince, hata olsa bile dosyanın kapanmasını. 5) int() ya da float() ile sayıya çevirmelisin.

Modül 7: 1) Sözdizimi hatası yazım kurallarına uymamaktır ve program çalışmadan bildirilir; mantık hatasında program çalışır ama yanlış sonuç üretir. 2) En alttaki satırdan; hata türü ve açıklaması oradadır. 3) ValueError. 4) Hata olsa da olmasa da her durumda. 5) Bütün hataları sessizce yutar; program yanlış çalışır ve neden bulunamaz.

Modül 8: 1) DNS. 2) 404 kaynağın bulunamadığını, 500 sunucuda beklenmeyen bir hata oluştuğunu anlatır; sunucu tarafı hatası 500'dür. 3) HTML içeriği ve yapıyı, CSS görünüşü, JavaScript davranışı belirler. 4) POST. 5) True ve None.

Modül 9: 1) Tablodaki her kaydı benzersiz biçimde tanımlayan sütun. 2) SELECT ad FROM musteriler WHERE sehir = 'Ankara'; 3) Siparişler tablosundaki bütün kayıtları siler. 4) IS NULL. 5) Parametrelili sorguyla: sorguda yer tutucu (soru işareti) kullanılır, değer ayrı argüman olarak verilir.

Modül 10: 1) git add değişiklikleri hazırlık alanına ekler; git commit onları mesajıyla birlikte kalıcı kayıt olarak geçmişe yazar. 2) Ana kodu bozmadan bağımsız çalışmak, değişikliği incelemeye açmak ve hazır olunca birleştirmek için. 3) Tek bir fonksiyonu ya da küçük bir kod parçasını, diğer parçalardan ayrı olarak. 4) Anahtar geçmişteki commit'te durur; anahtar hemen değiştirilmeli ve dosya .gitignore'a eklenmelidir. 5) Avantajlar: hızlı çözüm ve prototip, iş birimlerinin kendi süreçlerini düzenleyebilmesi. Sınırlar: karmaşık ihtiyaçlarda zorlanma, platforma bağımlılık, artan maliyet, verinin konumu (herhangi ikisi).

Sözlük

Algoritma: Bir problemi çözmek için izlenen, sırası belli, açık ve sonlu adımlar dizisi.

API (uygulama programlama arayüzü): Programların birbiriyle hangi kurallarla konuşacağını tanımlayan arayüz.

Back-end: Uygulamanın sunucu tarafı; iş kuralları, veri tabanı işlemleri ve diğer sistemlerle bağlantı.

Birim testi (unit test): Tek bir fonksiyonu ya da küçük bir kod parçasını ayrı olarak kontrol eden otomatik test.

Birincil anahtar (primary key): Bir tablodaki her kaydı benzersiz biçimde tanımlayan sütun.

Commit: Git'te belirli bir andaki değişikliklerin açıklayıcı bir mesajla kaydedilmiş hali.

CSS: Web sayfasının renk, yazı tipi, boşluk ve yerleşim gibi görünüş özelliklerini belirleyen dil.

Dal (branch): Ana koddan ayrılarak bağımsız olarak geliştirilen çalışma hattı.

Değişken: Bir değeri bellekte saklamak için kullanılan isim.

Döngü: Bir kod bloğunu belirli sayıda ya da bir koşul sağlandığı sürece tekrarlayan yapı (for, while).

Fonksiyon: Bir işi yapan, bir adla çağrılan, parametre alıp sonuç döndürebilen kod bloğu.

Front-end: Uygulamanın kullanıcının gördüğü ve etkileştiği kısmı.

Git: Dosyalardaki değişiklikleri kaydeden, dağıtık bir sürüm kontrol sistemi.

Hata ayıklama (debugging): Bir programdaki hatanın yerini bulup düzeltme süreci.

HTML: Web sayfasının içeriğini ve yapısını etiketlerle tanımlayan işaretleme dili.

HTTP ve HTTPS: İstemci ile sunucu arasındaki iletişimin kuralları; HTTPS, TLS ile şifrelenmiş halidir.

JavaScript: Web sayfalarına davranış ekleyen, tarayıcıda ve Node.js ile sunucuda çalışan programlama dili.

JSON: API'lerde yaygın kullanılan, insanların da okuyabildiği düz metin veri biçimi.

Modül: İçinde fonksiyon ve değişkenler bulunan, import ile başka programlarda kullanılabilen Python dosyası.

No-code ve low-code: Kod yazmadan ya da az kodla, görsel arayüzlerle uygulama ve iş akışı oluşturmayı sağlayan araçlar.

Parametre: Fonksiyon tanımında, dışarıdan alınacak değer için kullanılan ad.

Pull request: Bir daldaki değişikliklerin incelenip ana dala birleştirilmesi için açılan talep (GitLab'de merge request).

Sözde kod (pseudocode): Algoritmanın, belirli bir dilin kurallarına bağlı kalmadan kod benzeri sade cümlelerle yazılmış hali.

Sözlük (dictionary): Python'da anahtar-değer çiftlerinden oluşan veri yapısı.

SQL: İlişkisel veri tabanlarında veri okumak, eklemek, güncellemek ve silmek için kullanılan standart dil.

SQL enjeksiyonu: Kullanıcı girdisinin sorgu metnine doğrudan eklenmesiyle sorunun kötü niyetli biçimde değiştirilmesine yol açan güvenlik açığı.

Sürekli entegrasyon (CI): Kod depoya her gönderildiğinde testlerin otomatik olarak çalıştırılması uygulaması.

Veri tabanı: Verilerin düzenli, tutarlı ve güvenli biçimde saklandığı, bir yönetim sistemiyle erişilen yapı.

Yabancı anahtar (foreign key): Bir tablodaki sütunun başka bir tablonun birincil anahtarını göstererek iki tabloyu ilişkilendirmesi.

Sonraki adım

Eğitimi bitirdiysen bilginizi ölç. Daphne Kariyer'deki Yazılım Geliştirme, Python Programlama, Web Programlama, Front-End Developer, Back-End Developer, JavaScript ve SQL testleri ücretsizdir. Her test 20 sorudan oluşur; 55 ve üzeri puan alırsan isteğe bağlı olarak doğrulanabilir dijital sertifika alabilirsiniz.

daphnekariyer.com/dijital-yetkinlikler/