WEB & YAZILIM GELİŞTİRME SERİSİ · MODÜL 10

Python

Genel amaçlı programlama: sözdizimi, değişkenler, koşullar, döngüler, listeler, sözlükler ve fonksiyonlar; metinler, modüller, dosyalar ve hata yönetimi; OOP, JSON, sanal ortamlar; otomasyon, veri, web ve yapay zekâ. Özgün diyagramlar ve gerçek Python örnekleriyle.

Sözdizimi · Veri · OOP · Otomasyon · Eğitim amaçlıdır

Bu Kitap Hakkında

Bu modül, okunabilirliğiyle ünlü, genel amaçlı bir dil olan Python'u sıfırdan öğretir. PHP'den sonra ikinci sunucu/genel amaçlı dil olarak, aynı programlama kavramlarının farklı ve sade bir sözdizimiyle nasıl ifade edildiğini gösterir. Dört seviye ve yirmi altı bölüm boyunca print, değişkenler, operatörler, koşullar (girintiler), döngüler, listeler, sözlükler ve fonksiyonlardan; demet ve kümelere, metin işlemeye, liste üreticilerine, modüllere, dosyalara ve hata yönetimine; nesne yönelimli programlamaya, kalıtıma, veri modellemeye, JSON'a, sanal ortamlara ve kod düzenine (PEP 8); otomasyona, veri işlemeye, web ve API'lere, yapay zekâya ve güvenliğe kadar uzanır.

Her bölümde konuyu görselleştiren özgün bir diyagram (kod → terminal çıktısı panelleri, liste/sözlük yapı görselleri, sınıf kutuları, akış şemaları), çalıştırılabilir Python örnekleri, 'ne yazdırır?' kartı ve bir alıştırma yer alır. Güvenlik ve sorumlu kullanım baştan sona vurgulanır: girdi doğrulama, sırların gizlenmesi, bağımlılık güvenliği, tehlikeli işlevlerden (eval) kaçınma ve yapay zekâ söz konusu olduğunda halüsinasyon farkındalığı, veri gizliliği (KVKK) ve insan denetimi. Bu, on altı modüllük 'Web & Yazılım Geliştirme' serisinin onuncu modülüdür. Bu seri eğitim amaçlıdır.

Web & Yazılım Geliştirme Serisi · Modül 10

İçindekiler

SEVİYE 1

Python'a Giriş

Temeller: Python nedir, print ile ilk kod, değişkenler ve veri tipleri, operatörler, koşullar (girintiler), döngüler, listeler ve sözlükler, fonksiyonlar.

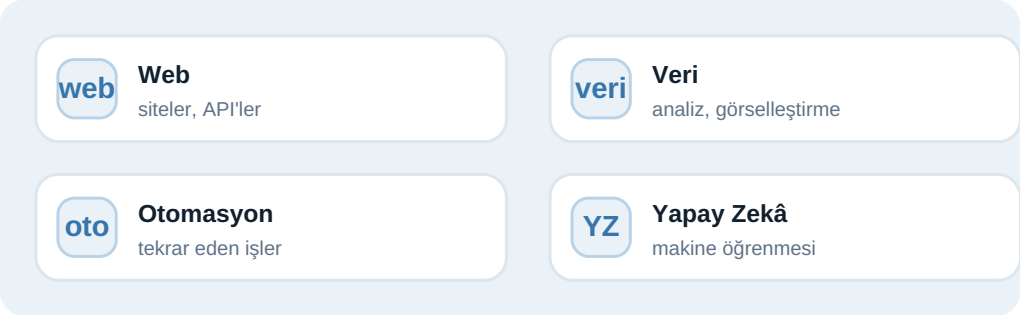
BÖLÜM 01

Python Nedir?

Python, okunabilirliğiyle ünlü, genel amaçlı bir programlama dilidir. Web, veri analizi, otomasyon ve yapay zekâ dahil çok geniş bir alanda kullanılır. Sade sözdizimi sayesinde başlangıç için en çok önerilen dillerden biridir.

Python neden bu kadar popüler?

- **Okunabilir:** sözdizimi neredeyse İngilizce gibi; az "gürültü".
- **Çok yönlü:** web, veri, otomasyon, yapay zekâ — hepsinde kullanılır.
- **Geniş ekosistem:** hazır kütüphanelerle çoğu işi kısa kodla yaparsın.
- **Yorumlanır:** kodu derlemeden, doğrudan çalıştırırsın.



Şema 1.1 — Python'un başlıca kullanım alanları: tek dil, çok amaç.

İPUCU

PHP'yi (Modül 9) bitirdiysen, Python'da çoğu kavramı tanıyacaksın: değişkenler, koşullar, döngüler, fonksiyonlar her dilde vardır — değişen yalnızca sözdizimidir. Python'un en belirgin farkı, kod bloklarını süslü parantez yerine **girintiyle (boşlukla)** ayırmasıdır; bu, kodu görsel olarak temiz tutar. Python iki ana sürümle anıldı (2 ve 3); bugün standart olan **Python 3**'tür ve bu modülde onu kullanıyoruz.

Ne yazdırır? ÇIKTI

Python kodu bir yorumlayıcı (interpreter) tarafından satır satır çalıştırılır; `print(...)` ile bir şey yazdırdığında sonucu doğrudan terminalde (veya çalıştırdığın ortamda) görürsün. Derleme adımı yoktur — yaz, çalıştır, sonucu gör.

Alıştırma

8 dk

Python'u tanı:

1. Python'un dört kullanım alanını ve birer örnek işi yaz.
2. Python'u başlangıç için cazip kılan iki özelliği belirt.
3. Python ile PHP'nin ortak yanı ne, farkı ne — kısaca yaz.

BÖLÜM 02

İlk Kod: print

Python'da ekrana bir şey yazdırmanın yolu print() fonksiyonudur. Çoğu dilde olduğu gibi, ilk programın bir şeyler yazdırmaktır. Python'da bu son derece sadedir — gereksiz hiçbir şey yok.

print() ile çıktı

- `print("metin")` — ekrana yazdırır.
- Tırnak içinde metin (string); tek veya çift tırnak.
- Noktalı virgül **gerekmez**; her satır bir ifadedir.

Şema 2.1 — print(), kodun çıktısını doğrudan terminale yazar.

İlk Python kodun

```
print("Merhaba, dünya!")
print("Python öğreniyorum.")
print(2 + 3)          # 5 yazdırır
```

İPUCU

Python'da satır sonuna noktalı virgül koymana **gerek yoktur** (PHP'den farklı olarak) — her satır kendi başına bir ifadedir. Yorum satırları `#` ile başlar ve çalıştırılmaz; kodu açıklamak için kullanılır. `print()` birden çok şeyi virgülle ayırarak yazdırabilir: `print("Yaş:", 30)` → "Yaş: 30". Bu sadelik, Python'un imzasıdır.

Ne yazdırır? ÇIKTI

`print("Merhaba, dünya!")` terminalde "Merhaba, dünya!" satırını gösterir. `print(2 + 3)` ise önce işlemi yapar, sonra sonucu (5) yazdırır. Her `print` çağrısı çıktıyı yeni bir satıra yazar.

◎ Alıştırma

10 dk

print kullan:

1. Kendini tanıtan üç satırlık bir Python kodu yaz.
2. Bir matematik işleminin sonucunu print ile yazdır.
3. Bir yorum satırı (#) ekleyip ne işe yaradığını yaz.

BÖLÜM 03

Değişkenler ve Veri Tipleri

Değişken, bir değeri saklayan adlandırılmış bir kutudur. Python'da değişken oluşturmak çok sadedir: bir isim, eşittir işareti ve değer. Tipi sen belirtmezsin; değer belirler (dinamik tipleme).

Değişken ve temel tipler

- `ad = "Ayşe"` — atama; `$` veya tip bildirimi yok.
- **int** (tam sayı), **float** (ondalık), **str** (metin), **bool** (True/False).
- `type(degisken)` ile tipini öğrenirsin.

Şema 3.1 — Değişkenler değer saklar; print birçok değeri birlikte yazdırır.

Değişkenler ve tipler

```
ad     = "Ayşe"      # str (metin)
yas    = 30          # int (tam sayı)
boy    = 1.68        # float (ondalık)
aktif  = True        # bool (True / False)
print(type(yas))     # <class 'int'>
```

İPUCU

Python **dinamik tiplidir**: bir değişkene önce metin, sonra sayı atayabilirsin (tavsiye edilmese de). Mantıksal değerler Python'da büyük harfle başlar: `True` ve `False`. Anlamlı değişken isimleri seçmek (Modül 6'daki gibi) kodu okunur kılar; Python geleneğinde isimler `kucuk_harf_ve_alt_tire` biçimindedir (snake_case). PHP'deki `$` işareti Python'da **yoktur** — sadece ismi yazarsın.

Ne yazdırır? ÇIKTI

`ad = "Ayşe"` ve `yas = 30` tanımlandıktan sonra `print(ad, "—", yas, "yasinda")` bunları tek satırda birleştirip "Ayşe — 30 yasinda" yazdırır. `type()` ise bir değişkenin hangi tipte olduğunu söyler.

◎ Alıştırma

10 dk

Değişken kullan:

1. Ad, şehir ve yaş için üç değişken tanımla ve print ile yazdır.
2. type() ile bir sayı ve bir metnin tipini öğren.
3. snake_case ile anlamlı iki değişken adı yaz.

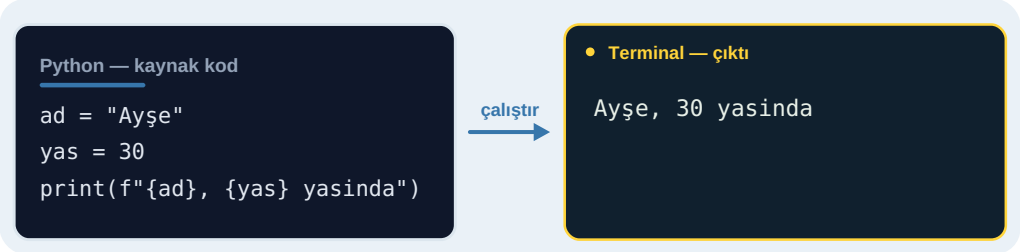
BÖLÜM 04

Operatörler ve İfadeler

Operatörler, değerler üzerinde işlem yapmanı sağlar. Python'da metinleri birleştirmenin en şık yolu ise f-string'lerdir: değişkenleri doğrudan metnin içine gömersin.

Operatörler ve f-string

- Aritmetik: `+ - * / // % **` (// tam bölme, ** üs).
- Karşılaştırma: `== != > < >= <=`.
- Metin birleştirme: `+` veya (tercihen) **f-string**.
- f-string: `f"Merhaba {ad}"` — değişkeni süslü parantezde göm.



Şema 4.1 — f-string: değişkenleri doğrudan metnin içine yerleştirir.

Operatörler ve f-string

```
 a = 10
b = 3
print(a + b)        # 13
print(a / b)        # 3.333...
print(a // b)       # 3   (tam bölme)
print(a ** 2)       # 100 (üs)
print(f"{a} ve {b}")  # 10 ve 3
```

İPUCU

f-string'ler (formatlanmış metin) Python'da metin oluşturmanın en okunur yoludur: metni `f` ile başlatır, değişkenleri `{ }` içine yazarsın. PHP'nin çift tırnaklı değişken çözmesine benzer ama daha güçlüdür: `{ }` içinde işlem bile yapabilirsin: `f"Toplam: {a + b}"`. Bölme operatörlerine dikkat: `/` her zaman ondalık döndürür (10/2 → 5.0), `//` ise tam sayı bölme yapar.

Ne yazdırır? ÇIKTI

`f"{ad}, {yas} yasinda"` ifadesi, `{ad}` ve `{yas}` yerine değişkenlerin değerlerini koyarak "Ayşe, 30 yasinda" üretir. f-string'ler, dinamik metinler oluşturmanın en temiz yoludur — değişken ve metni iç içe, okunur biçimde yazarsın.

Alıştırma 10 dk

Operatör ve f-string:

1. İki sayının toplamını, bölümünü ve tam bölümünü yazdır.
2. Bir f-string ile ad ve yaşı tek cümlede yazdır.
3. / ile // arasındaki farkı bir örnekle göster.

BÖLÜM 05

Koşullar: if / elif / else

Programların karar vermesini koşullar sağlar. Python'da if bir koşulu sınar; elif başka bir koşulu; else ise hiçbiri tutmadığında çalışır. Python'ın en belirgin özelliği burada ortaya çıkar: bloklar girintiyle (boşlukla) tanımlanır.

if / elif / else ve girinti

- `if koşul:` — iki nokta üst üste ile biter.
- Bloğun içeriği **girintilidir** (genelde 4 boşluk).
- `elif` (else if) ve `else` ek dallardır.

Şema 5.1 — if/else: koşula göre girintili bloklardan biri çalışır.

if / elif / else

```
not_ = 75
if not_ >= 85:
    print("Pekiyi")
elif not_ >= 60:
    print("Geçer")
else:
    print("Kaldı")
```

İPUCU

Python'da **girinti zorunludur ve anlamlıdır** — süslü parantez yoktur. Bir bloğun neresinin `if`'e ait olduğunu girinti belirler. Tutarlı ol: tüm proje boyunca 4 boşluk kullan (sekme ile boşluğu karıştırma — en sık girinti hatası budur). Koşullar sırayla denenir; ilk doğru olan dal çalışır ve gerisi atlanır. Mantıksal operatörler kelime hâlindedir: `and`, `or`, `not` (PHP'deki `&&` / `||` yerine).

Ne yazdırır? ÇIKTI

`not_ = 75` için kod sırayla bakar: 85'ten büyük mü (hayır), 60'tan büyük mü (evet) → "Geçer" yazdırılır ve `else`'e bakılmaz. Girintili satır, o koşula ait olan koddur; girintiyi değiştirirsen kodun anlamı değişir.

Alıştırma 12 dk

Koşul yaz:

1. Bir sayının pozitif/negatif/sıfır olduğunu yazdıran if/elif/else yaz.
2. and kullanarak iki koşulu birleştir.
3. Girintiyi bozarsan ne olur, dene ve gözlemle.

BÖLÜM 06

Döngüler: for / while

Döngüler, bir işi tekrar tekrar yapmanı sağlar. Python'da for döngüsü bir dizi (liste, aralık, metin) üzerinde gezinir; while ise bir koşul doğru olduğu sürece çalışır. range() ile sayı aralıkları üretirsin.

for ve while

- `for x in dizi:` — her eleman için tekrar.
- `range(1, 4)` — 1, 2, 3 sayılarını üretir.
- `while koşul:` — koşul doğru oldukça çalışır.

Şema 6.1 — for + range(): belirli sayıda tekrar eden döngü.

for ve while

```
# for: bir aralık üzerinde
for i in range(1, 4):
    print(i)            # 1, 2, 3

# for: bir liste üzerinde
for renk in ["kırmızı", "yeşil"]:
    print(renk)

# while: koşul sürdükçe
sayac = 0
while sayac < 3:
    sayac += 1
```

İPUCU

Python'ın `for` döngüsü çoğu dilden farklıdır: bir sayaç yönetmek yerine, doğrudan bir **dizinin elemanları üzerinde** gezinir (`for renk in renkler`). Bu, kodu çok okunur kılar. Sayı aralığı gerekiyorsa `range()` kullan: `range(5)` → 0,1,2,3,4 (son sayı dahil değildir). `while` kullanırken koşulun bir gün yanlış olacağından emin ol — yoksa sonsuz döngüye girersin.

Ne yazdırır? ÇIKTI

`for i in range(1, 4)` döngüsü i'yi sırayla 1, 2, 3 yapar (4 dahil değil) ve her turda "Satır {i}" yazdırır. Sonuç üç satırlık bir çıktıdır. range(), belirli sayıda tekrar için en yaygın yoldur.

Alıştırma 12 dk

Döngü yaz:

1. range() ile 1'den 10'a sayıları yazdıran for döngüsü yaz.
2. Bir liste üzerinde gezinip her elemanı yazdır.
3. range(5)'in neden 5'i içermediğini açıkla.

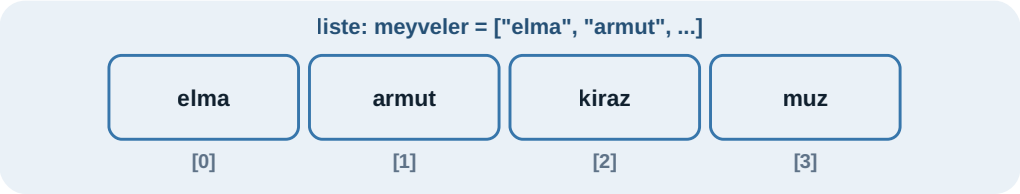
BÖLÜM 07

Listeler ve Sözlükler

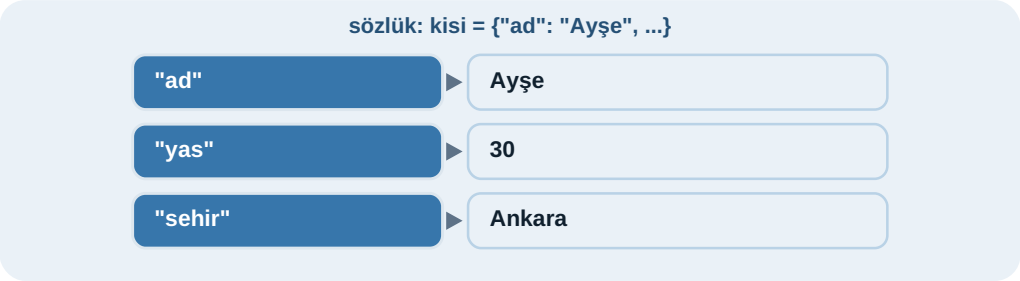
Python'ın en güçlü iki veri yapısı listeler ve sözlüklerdir. Liste, sıralı bir değerler dizisidir (indeksle erişilir). Sözlük (dict) ise anahtar–değer çiftlerinden oluşur (anlamlı bir anahtarla erişilir).

Liste ve sözlük

- **Liste:** `["a", "b", "c"]` — sıralı; `liste[0]` ile erişim.
- **Sözlük:** `{"ad": "Ayşe"}` — anahtar–değer; `d["ad"]` ile erişim.
- Listeye `.append()` ile eleman eklenir.



Şema 7.1 — Liste: sıralı, 0'dan başlayan indekslerle erişilir.



Şema 7.2 — Sözlük: her değere anlamlı bir anahtarla erişilir.

Listeler ve sözlükler

```
 # Liste
meyveler = ["elma", "armut", "kiraz"]
print(meyveler[0])        # elma
meyveler.append("muz")    # listeye ekle

# Sözlük (dict)
kisi = {"ad": "Ayşe", "yas": 30}
print(kisi["ad"])         # Ayşe
```

İPUCU

Liste ile sözlük arasındaki seçim, "sıraya mı yoksa anlama mı göre erişeceğim?" sorusuna bağlıdır. Bir **liste** sıralı şeyler içindir (yapılacaklar, ürünler); **sözlük** ise bir şeyin özellikleri içindir (bir kullanıcının adı, yaşı, şehri). Sözlükler PHP'deki ilişkisel dizilerin Python karşılığıdır ve Modül 8'deki veritabanı satırları Python'a genelde sözlük olarak gelir. İkisini iç içe kullanmak çok yaygındır: sözlüklerden oluşan bir liste (kayıt listesi).

Ne yazdırır?

ÇIKTI

`meyveler[0]` ilk meyveyi ("elma") döner; `kisi["ad"]` ise "ad" anahtarının değerini ("Ayşe") döner. Listede konuma (indeks), sözlükte anlamlı anahtara göre erişirsin. `.append()` listeye yeni eleman ekler.

Alıştırma

14 dk

Veri yapısı kullan:

1. Beş şehirden oluşan bir liste oluştur; üçüncüsünü yazdır ve birini ekle.
2. Bir kişiyi tanımlayan sözlük oluştur (ad, yas, sehir).
3. Ne zaman liste, ne zaman sözlük kullanırsın — birer örnekle yaz.

BÖLÜM 08

Fonksiyonlar

Fonksiyon, bir işi yapan, isimlendirilmiş ve tekrar kullanılabilir bir kod bloğudur. Python'da fonksiyonlar def anahtar kelimesiyle tanımlanır. Bir kez yazar, defalarca çağırırsın.

def ile fonksiyon

- `def ad(parametreler):` ile tanımlanır.
- Gövdesi girintilidir; `return` ile değer döndürür.
- Çağrı: `ad(argümanlar)` .

Şema 8.1 — Fonksiyon: parametre alır, iş yapar, değer döndürür.

Fonksiyon tanımı

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

print(topla(3, 5))       # 8
print(topla(10, 20))     # 30  (tekrar kullan)

# Varsayılan değerli parametre
def selamla(isim="misafir"):
    return f"Merhaba, {isim}"
```

İPUCU

Fonksiyonlar, Modül 6'daki "problemi parçalara bölme" ve "soyutlama" ilkelerinin somut hâlidir. İyi bir fonksiyon **tek bir işi** iyi yapar ve adı ne yaptığını söyler. Parametrelere varsayılan değer verebilirsin (`isim="misafir"`), böylece argümansız da çağrılabilir. `return` bir değer geri döndürür; eğer `return` yazmazsan fonksiyon `None` döndürür. `print` ile `return` farkını unutma: biri yazdırır, diğeri değeri geri verir (sonra kullanmak üzere).

Ne yazdırır? ÇIKTI

`selamla("Ayşe")` çağrısı fonksiyonu çalıştırır, "Merhaba, Ayşe!" metnini `return` ile döndürür; `print` bunu terminalde gösterir. Aynı fonksiyonu farklı argümanlarla tekrar çağırarak kodu bir kez yazıp defalarca kullanırsın.

Alıştırma 12 dk

Fonksiyon yaz:

1. İki sayının çarpımını döndüren bir fonksiyon yaz ve çağır.
2. Bir ismi alıp selamlama döndüren fonksiyon yaz.
3. Bir parametreye varsayılan değer ver ve argümansız çağır.

SEVİYE 2

Veri ve Yapılar

Python'un gücü veride: demetler ve kümeler, metinlerle çalışmak, liste üreticileri, modüller ve kütüphaneler, dosyalarla çalışmak ve hata yönetimi.

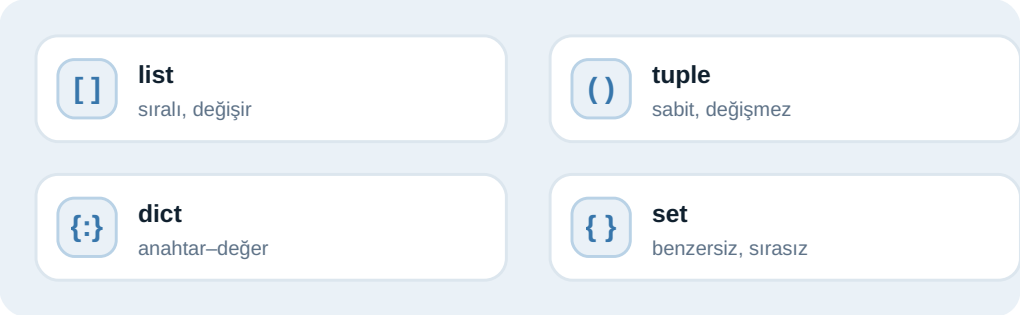
BÖLÜM 09

Daha Fazla Veri Yapısı

Liste ve sözlüğün yanında Python iki yapı daha sunar: demet (tuple) ve küme (set). Her birinin kendine özgü bir amacı vardır; doğru yapıyı seçmek kodu hem net hem verimli kılar.

Dört temel veri yapısı

- **list** `[ ]` — sıralı, değiştirilebilir.
- **tuple** `( )` — sıralı, **değiştirilemez** (sabit).
- **dict** `{anahtar: değer}` — anahtar–değer.
- **set** `{ }` — benzersiz, sırasız (tekrar yok).



Şema 9.1 — Python'un dört temel veri yapısı ve özellikleri.

tuple ve set

```
# tuple: değiştirilemez (örn. koordinat)
nokta = (3, 5)
print(nokta[0])          # 3

# set: benzersiz değerler
sehirler = {"Ankara", "İzmir", "Ankara"}
print(sehirler)          # {'Ankara', 'İzmir'} (tekrar yok)
```

İPUCU

Doğru yapıyı amaca göre seç: değişmemesi gereken bir grup değer (örn. bir koordinat, sabit ayarlar) için **tuple**; tekrarları otomatik elemek veya "bu eleman var mı?" diye hızlı bakmak için **set**; sıralı ve değişebilir bir koleksiyon için **list**; anlamlı anahtarlarla erişim için **dict**. Bir listeden tekrarları silmenin en kısa yolu da set kullanmaktır: `list(set(liste))` . Doğru yapı, kodu hem okunur hem hızlı yapar.

Ne yazdırır? ÇIKTI

`nokta = (3, 5)` bir demettir ve değiştirilemez — `nokta[0] = 9` denersen hata alırsın. `sehirler` kümesi ise "Ankara"yı iki kez yazsan bile tek tutar; çıktıda her şehir yalnızca bir kez görünür. Her yapı, farklı bir ihtiyaca cevap verir.

Alıştırma 12 dk

Yapı seç:

1. Bir koordinat için neden tuple kullanılır, açıkla.
2. Bir listeden tekrarları set ile temizleyen kod yaz.
3. Dört yapıyı ve birer kullanım örneğini yaz.

BÖLÜM 10

Metinlerle Çalışmak (String)

Metin işleme, çoğu programın günlük işidir: kullanıcı girdisini temizlemek, biçimlendirmek, parçalamak. Python, metinlerle çalışmak için zengin ve okunur metotlar sunar.

Sık kullanılan metin metotları

- `.upper()` / `.lower()` — büyük/küçük harf.
- `.strip()` — baştaki/sondaki boşlukları kırp.
- `.split(",")` — metni parçalara böl (liste döner).
- `.replace("a","b")` — değiştir; `len(metin)` — uzunluk.

Şema 10.1 — Metin metotları zincirlenebilir: önce kırp, sonra büyüt.

Metin işleme

```
 metin = "elma,armut,kiraz"
print(metin.split(","))      # ['elma', 'armut', 'kiraz']
print(metin.replace("a", "A"))
print(len(metin))            # 16

# dilimleme (slicing): bir parçasını al
print(metin[0:4])            # elma
```

İPUCU

Metin metotları **zincirlenebilir**: `ad.strip().upper()` önce boşlukları kırpar, sonra büyük harfe çevirir. **Dilimleme** (slicing) Python'un güçlü bir özelliğidir: `metin[0:4]` ilk dört karakteri alır (4 dahil değil); `metin[-1]` son karakteri verir. Unutma: Python'da metinler **değiştirilemez** — bir metot yeni bir metin döndürür, orijinali değiştirmez. Kullanıcı girdisini işlerken `.strip()` ile baştaki/sondaki boşlukları temizlemek yaygın bir ilk adımdır.

Ne yazdırır? ÇIKTI

`" Ayşe Yılmaz ".strip().upper()` önce baştaki ve sondaki boşlukları atar, sonra tümünü büyük harfe çevirir → "AYŞE YILMAZ". `.split(",")` ise virgülle ayrılmış bir metni bir listeye böler. Metin metotları, ham veriyi düzenli hâle getirmenin temel araçlarıdır.

Alıştırma 12 dk

Metin işle:

1. Bir metni büyük harfe çevir ve boşluklarını kırp.
2. Virgülle ayrılmış bir metni split ile listeye böl.
3. Dilimleme ile bir metnin ilk üç karakterini al.

BÖLÜM 11

Liste Üreticileri (Comprehensions)

Python'un en sevilen özelliklerinden biri liste üreticileridir (list comprehension): bir listeyi tek, okunur bir satırda dönüştürmeni veya süzmeni sağlar. Klasik bir döngüyle yazılabilecek çok şeyi kısaltır.

Comprehension mantığı

- `[ifade for x in dizi]` — her elemanı dönüştür.
- `[x for x in dizi if koşul]` — koşula göre süz.
- Sonuç her zaman yeni bir listedir.

Şema 11.1 — Comprehension: aynı işi daha kısa ve okunur yazar.

Liste üreticileri

```
 # Dönüştür: her sayının karesi
kareler = [x*x for x in range(1, 6)]
print(kareler)          # [1, 4, 9, 16, 25]

# Süz: yalnızca çift sayılar
ciftler = [x for x in range(10) if x % 2 == 0]
print(ciftler)          # [0, 2, 4, 6, 8]
```

İPUCU

Liste üreticileri, Python'a özgü güçlü ve okunur bir kalıptır — ama abartma: çok karmaşık (iç içe, uzun koşullu) comprehension'lar okunmaz hâle gelir; o noktada klasik döngü daha nettir. `enumerate(dizi)` ile hem indeksi hem değeri (`for i, x in enumerate(liste)`), `zip(a, b)` ile iki listeyi birlikte dolaşabilirsin — bunlar Python'da döngülerin sık kullanılan yardımcılarıdır. Okunabilirlik her zaman önceliklidir.

Ne yazdırır? ÇIKTI

`[x*x for x in range(1, 6)]` 1'den 5'e her sayının karesini alıp yeni bir liste oluşturur: `[1, 4, 9, 16, 25]`. `if` ekleyerek süzersin: yalnızca çift sayılar. Tek satırda, bir döngünün yaptığı işi okunur biçimde ifade edersin.

Alıştırma 12 dk

Comprehension yaz:

1. 1'den 10'a sayıların karelerini comprehension ile üret.
2. Bir listeden yalnızca 100'den büyük sayıları süz.
3. Aynı işi klasik döngüyle yazıp karşılaştır.

BÖLÜM 12

Modüller ve Kütüphaneler

Her şeyi sıfırdan yazmana gerek yok. Python'ın zengin standart kütüphanesi ve on binlerce dış paketi, çoğu işi hazır sunar. import ile bunları kodunla buluşturursun.

import ve paketler

- `import math` — bir modülü içe aktar.
- `from datetime import date` — belirli bir parçayı al.
- **Standart kütüphane:** Python'la birlikte gelir (math, random, json...).
- **Dış paketler:** `pip install ...` ile kurulur.

Şema 12.1 — Hazır kod: standart kütüphane + pip paketleri import ile gelir.

Modül kullanımı

```
 import math
print(math.sqrt(16))        # 4.0

from random import randint
print(randint(1, 6))        # 1–6 arası rastgele

import json                 # JSON ile çalışmak için
```

İPUCU

Python'un "piller dahil" (batteries included) felsefesi vardır: standart kütüphane (math, random, json, os, datetime...) o kadar zengindir ki çoğu temel işi paket kurmadan yaparsın. Daha özel işler için **pip** (Python'un paket yöneticisi) ile dış paket kurarsın — ama her paketi körü körüne kurma: çok sayıda bağımlılık hem güvenlik riski hem bakım yüküdür. Tanınmış, bakımı sürdürülen paketleri tercih et ve bir sonraki seviyede göreceğin **sanal ortamlarla** bağımlılıkları izole et.

Ne yazdırır? ÇIKTI

`import math` sonrası `math.sqrt(16)` 4.0 döner; `from random import randint` ile `randint(1, 6)` 1–6 arası rastgele bir sayı verir. import, başkalarının yazıp test ettiği hazır kodu kendi programında kullanmanın yoludur — tekerleği yeniden icat etmezsin.

Alıştırma 10 dk

Modül kullan:

1. math modülünü import edip bir karekök hesapla.
2. random ile 1–100 arası rastgele bir sayı üret.
3. Standart kütüphane ile dış paket arasındaki farkı açıkla.

BÖLÜM 13

Dosyalarla Çalışmak

Programlar çoğu zaman kalıcı veriyle çalışır: bir dosyadan okur, sonucu bir dosyaya yazar. Python'da dosya işlemleri sade ve güvenlidir — özellikle with bloğuyla.

Dosya okuma ve yazma

- `with open("dosya.txt") as f:` — güvenli açma.
- `f.read()` okur, `f.write(...)` yazar.
- `with` bloğu dosyayı otomatik kapatır.

Şema 13.1 — with bloğu: dosyayı açar, işini yapar, otomatik kapatır.

Dosya okuma/yazma

```
 # Yazma ("w" = write, üzerine yazar)
with open("notlar.txt", "w") as f:
    f.write("İlk notum\n")

# Okuma ("r" = read)
with open("notlar.txt", "r") as f:
    icerik = f.read()
    print(icerik)
```

İPUCU

Her zaman `with` bloğu kullan. `with open(...) as f:` dosyayı açar ve blok bittiğinde (hata olsa bile) otomatik kapatır — açık kalan dosyalar kaynak sızıntısına ve veri kaybına yol açar. Dosya kipine dikkat: `"w"` dosyanın üzerine yazar (var olan içerik silinir!), `"a"` sonuna ekler, `"r"` okur. Türkçe karakterler için dosyayı `encoding="utf-8"` ile açmak iyi bir alışkanlıktır: `open("d.txt", "r", encoding="utf-8")`.

Ne yazdırır? ÇIKTI

İlk blok "notlar.txt" dosyasına bir satır yazar (yoksa oluşturur); ikinci blok aynı dosyayı okuyup içeriğini ekrana basar. `with` sayesinde dosyayı elle kapatmana gerek kalmaz — blok bittiğinde güvenle kapanır. Çıktı, dosyaya yazdığın metindir.

Alıştırma 12 dk

Dosya kullan:

1. Bir dosyaya üç satır yazan kod yaz (with kullan).
2. Aynı dosyayı okuyup ekrana basan kod yaz.
3. "w" ile "a" kipinin farkını açıkla.

BÖLÜM 14

Hata Yönetimi

Hatalar kaçınılmazdır: dosya bulunamaz, kullanıcı sayı yerine harf girer, bir bölme sıfıra düşer. İyi bir program hataları yakalar ve çökmek yerine nazikçe baş eder. Python'da bunun yolu try/except'tir.

try / except

- `try:` — riskli kodu dene.
- `except HataTipi:` — o hata olursa yakala.
- Program çökmez; nazikçe devam eder.

Şema 14.1 — try/except: hatayı yakala, çökmeyi önle, nazikçe devam et.

Hata yakalama

```
try:
    yas = int(input("Yaş: "))
    print(f"Gelecek yıl {yas + 1} olacaksın")
except ValueError:
    print("Lütfen geçerli bir sayı girin.")
```

İPUCU

Belirli hata tiplerini yakala (`except ValueError` , `except FileNotFoundError`) — her şeyi yutan çıplak `except:` kullanmak, gerçek hataları gizleyip hata ayıklamayı zorlaştırır. Yakaladığın hatayla anlamlı bir şey yap: kullanıcıyı bilgilendir, varsayılan bir değer kullan veya logla. `try` bloğunu olabildiğince **dar** tut — yalnızca gerçekten hata fırlatabilecek satırı sar. Bu yaklaşım, programını beklenmedik girdilere karşı dayanıklı kılar.

Ne yazdırır? ÇIKTI

Kullanıcı sayı yerine harf girerse `int(...)` bir `ValueError` fırlatır; `except ValueError` bunu yakalar ve "Lütfen geçerli bir sayı girin." mesajını gösterir — program çökmez. Geçerli bir sayı girilirse `try` bloğu normal çalışır ve sonucu yazdırır.

Alıştırma 12 dk

Hata yönet:

1. Kullanıcı girdisini sayıya çeviren, hatayı yakalayan kod yaz.
2. Sıfıra bölme hatasını (ZeroDivisionError) yakala.
3. Çıplak except: kullanmanın neden sakıncalı olduğunu açıkla.

SEVİYE 3

Programı Yapılandırmak

Daha büyük programlar: nesne yönelimli programlama, kalıtım, sözlüklerle veri modellemek, JSON, sanal ortamlar ve paketler, kod düzeni (PEP 8).

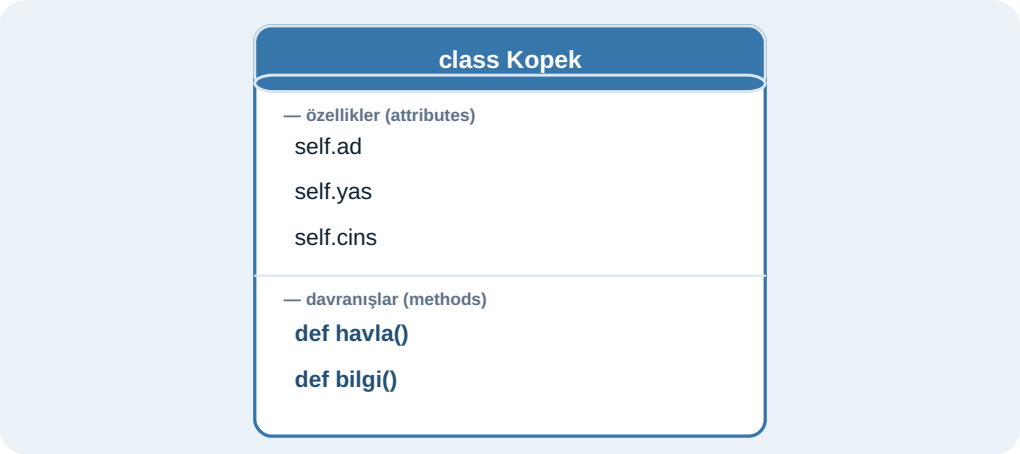
BÖLÜM 15

Nesne Yönelimli Programlama (OOP)

Programlar büyüdükçe, ilişkili veri ve davranışı bir arada paketlemek işe yarar. Nesne yönelimli programlama (OOP), bunu sınıflarla yapar: bir sınıf, özellikleri (veri) ve davranışları (metotlar) tek bir kalıpta birleştirir.

Sınıf ve nesne

- `class Ad:` — bir kalıp tanımlar.
- `__init__` — nesne oluşturulurken çalışan kurucu.
- `self` — nesnenin kendisine işaret eder.



Şema 15.1 — Bir sınıf: özellikler (veri) ve davranışlar (metotlar) bir arada.

Sınıf tanımı ve kullanımı

```
class Kopek:
    def __init__(self, ad, yas):
        self.ad = ad
        self.yas = yas
    def havla(self):
        return f"{self.ad}: Hav!"

kopek = Kopek("Karabaş", 3)
print(kopek.havla())         # Karabaş: Hav!
```

İPUCU

OOP, Modül 6'daki **soyutlama** ve "ilgili şeyleri bir arada tutma" fikirlerinin güçlü bir biçimidir. `__init__` metodu, her yeni nesne oluşturulduğunda otomatik çalışır ve nesnenin başlangıç özelliklerini ayarlar. `self`, her metodun ilk parametresidir ve "bu nesnenin kendisi" demektir — nesnenin özelliklerine `self.ad` ile erişirsin. PHP'deki OOP'ye (`$this`) çok benzer; Python'da `$this` yerine `self` kullanılır. Küçük betikler için fonksiyonlar yeterli olabilir; OOP büyük, yapılı programlarda parlar.

Ne yazdırır? ÇIKTI

`Kopek("Karabaş", 3)`, sınıf kalıbından somut bir nesne üretir; `__init__` ad ve yaşı ayarlar. `kopek.havla()` o nesnenin davranışını çağırır ve "Karabaş: Hav!" döner. Sınıfı bir kez tanımlar, ondan istediğin kadar nesne (farklı köpekler) üretirsin.

Alıştırma 14 dk

Sınıf yaz:

1. Bir "Araba" sınıfı tasarla: özellikleri (marka, hiz) ve bir metodu.
2. __init__ ile başlangıç değerlerini ayarla.
3. Sınıftan iki nesne oluştur ve bir metodunu çağır.

BÖLÜM 16

Sınıflar Arası İlişkiler: Kalıtım

Kalıtım (inheritance), bir sınıfın başka bir sınıfın özelliklerini ve davranışlarını devralmasıdır. Ortak özellikleri bir üst sınıfta toplar, özel olanları alt sınıflarda eklersin — tekrarı önlersin.

Üst ve alt sınıf

- **Üst sınıf:** ortak özellik/davranışlar (örn. Hayvan).
- **Alt sınıf:** üstü devralır + kendine özel ekler (örn. Kopek).
- `class Kopek(Hayvan):` — Hayvan'dan miras alır.

Şema 16.1 — Kalıtım: ortak özellikler üst sınıfta, özel olanlar alt sınıflarda.

Kalıtım

```
class Hayvan:
    def __init__(self, ad):
        self.ad = ad
    def bilgi(self):
        return f"Ben {self.ad}"

class Kopek(Hayvan):        # Hayvan'dan miras alır
    def havla(self):
        return "Hav!"

k = Kopek("Karabaş")
print(k.bilgi())            # Ben Karabaş (devralındı)
```

İPUCU

Kalıtım, "bir X bir Y türüdür" ilişkisini ifade eder: bir köpek bir hayvandır, bu yüzden `Kopek(Hayvan)` doğaldır. Üst sınıftaki kod alt sınıflarda **tekrar yazılmadan** kullanılır — kalıtımın asıl faydası budur. Ama abartma: çok derin kalıtım zincirleri (sınıfın sınıfının sınıfı...) kodu izlemeyi zorlaştırır. Modern tasarımda çoğu zaman "kompozisyon" (bir nesnenin başka nesneleri içermesi) kalıtıma tercih edilir. Basit tut: gerçekten ortak olan davranışları paylaş.

Ne yazdırır? ÇIKTI

`Kopek` sınıfı `Hayvan`'dan miras aldığı için, `bilgi()` metodunu yeniden yazmadan kullanır: `k.bilgi()` "Ben Karabaş" döner. `havla()` ise yalnızca `Kopek`'e özeldir. Ortak olan üstte tanımlanır, özel olan altta eklenir — kod tekrarı önlenir.

Alıştırma 14 dk

Kalıtım kur:

1. Bir "Hayvan" üst sınıfı ve ondan türeyen bir alt sınıf yaz.
2. Üst sınıftaki bir metodun alt sınıfta nasıl kullanıldığını göster.
3. Kalıtımın kod tekrarını nasıl önlediğini açıkla.

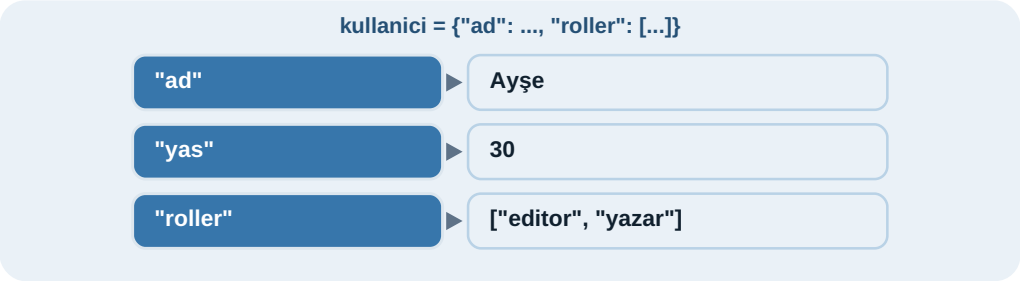
BÖLÜM 17

Sözlüklerle Veri Modellemek

Gerçek dünyadaki varlıkları (bir kullanıcı, bir ürün, bir sipariş) programda temsil etmenin en yaygın yolu sözlüklerdir. Sözlükleri iç içe ve listelerle birleştirerek karmaşık veriyi düzenli biçimde modellersin.

Sözlüklerle modelleme

- Bir varlık = bir sözlük (anahtarlar = özellikler).
- Çok varlık = sözlüklerden oluşan bir liste.
- İç içe sözlükler = karmaşık, hiyerarşik veri.



Şema 17.1 — Bir varlık, anahtar–değer çiftleriyle (iç içe yapılarla) modellenir.

Veri modelleme

```
 # Tek varlık
kullanici = {"ad": "Ayşe", "yas": 30}

# Çok varlık: sözlük listesi
kullanicilar = [
    {"ad": "Ayşe", "sehir": "Ankara"},
    {"ad": "Veli", "sehir": "İzmir"},
]
for k in kullanicilar:
    print(k["ad"], "—", k["sehir"])
```

İPUCU

Bu kalıp — **sözlüklerden oluşan bir liste** — Python'da veriyle çalışmanın belkemiğidir. Modül 8'deki bir veritabanı sorgusu Python'a tam olarak böyle döner: her satır bir sözlük, tüm sonuç bir liste. Bir API'den gelen JSON verisi de (sonraki bölüm) genelde bu yapıdadır. `for` ile listeyi dolaşıp her sözlüğün anahtarlarına erişerek veriyi işlersin. İç içe yapıları okurken anahtar adlarını dikkatli yaz — var olmayan bir anahtar `KeyError` verir; `.get("anahtar", varsayilan)` ile güvenli erişebilirsin.

Ne yazdırır? ÇIKTI

`kullanicilar`, her biri bir kişiyi temsil eden sözlüklerden oluşan bir listedir. `for` döngüsü her sözlüğü dolaşır ve `k["ad"]`, `k["sehir"]` ile değerlerine erişip yazdırır. Bu yapı, bir veritabanından veya API'den gelen veriyi temsil etmenin en doğal yoludur.

Alıştırma 14 dk

Veri modelle:

1. Üç ürünü, sözlük listesi olarak modelle (ad, fiyat).
2. Listeyi dolaşıp her ürünün adını ve fiyatını yazdır.
3. .get() ile var olmayan bir anahtara güvenli erişimi göster.

BÖLÜM 18

JSON ve Veri Değişimi

Farklı sistemler arasında veri paylaşmanın ortak dili JSON'dur (Backend modülünde görmüştün). Python sözlükleri ile JSON neredeyse aynı yapıdadır; json modülü ikisi arasında kolayca çeviri yapar.

Python ↔ JSON

- `json.dumps(sozluk)` — Python'dan JSON metnine.
- `json.loads(metin)` — JSON metninden Python'a.
- JSON, API'ler ve dosyalar arası veri taşır.

Şema 18.1 — JSON: Python verisini sistemler arası taşınabilir metne çevirir.

JSON ile çalışmak

```
import json

kisi = {"ad": "Ayşe", "yas": 30}
metin = json.dumps(kisi)     # '{"ad": "Ayşe", "yas": 30}'

# Geri çevir
geri = json.loads(metin)
print(geri["ad"])            # Ayşe
```

İPUCU

JSON'un Python'da bu kadar doğal olmasının nedeni, yapısının sözlük ve listelerle birebir örtüşmesidir: JSON nesnesi → Python sözlüğü, JSON dizisi → Python listesi. Bu yüzden bir API'den (Modül 7) JSON çekmek ve onu Python'da işlemek çok kolaydır. Türkçe karakterleri korumak için `json.dumps(veri, ensure_ascii=False)` kullan. JSON dosyaya yazarken/okurken `json.dump` / `json.load` (sondaki "s" olmadan) dosya nesnesiyle çalışır.

Ne yazdırır? ÇIKTI

`json.dumps(kisi)` Python sözlüğünü bir JSON metnine çevirir (bir API'ye göndermeye veya dosyaya yazmaya hazır); `json.loads(metin)` ise bu metni tekrar bir Python sözlüğüne dönüştürür. Böylece `geri["ad"]` ile değerlere yeniden erişebilirsin. JSON, veriyi sistemler arasında taşımanın ortak dilidir.

Alıştırma 12 dk

JSON kullan:

1. Bir sözlüğü json.dumps ile JSON metnine çevir.
2. Bir JSON metnini json.loads ile sözlüğe çevirip bir değere eriş.
3. JSON ile Python sözlüğünün neden bu kadar benzer olduğunu açıkla.

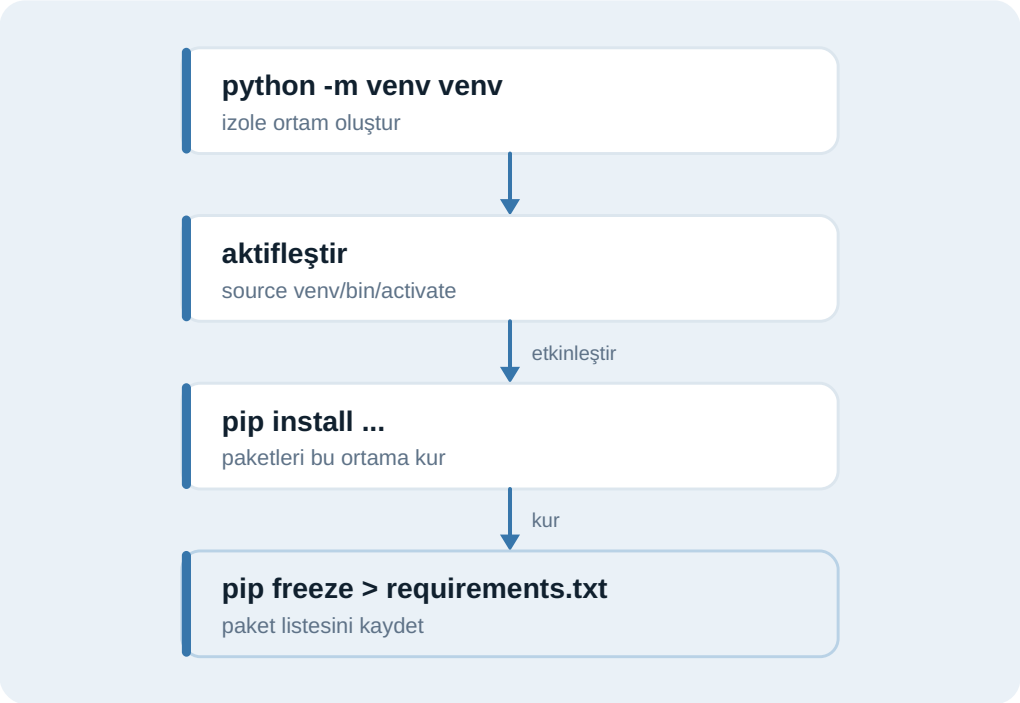
BÖLÜM 19

Sanal Ortamlar ve Paketler

Farklı projeler farklı paket sürümleri gerektirebilir. Sanal ortam (virtual environment), her projeye kendi izole paket alanını verir — böylece projeler birbirini bozmaz. Bu, profesyonel Python çalışmasının standardıdır.

venv ve pip

- `python -m venv venv` — sanal ortam oluştur.
- Aktifleştir, sonra `pip install paket` ile kur.
- `requirements.txt` — projenin paket listesi.



Şema 19.1 — Sanal ortam: her projeye kendi izole paket alanı.

Sanal ortam akışı (terminal)

```
 # 1. Ortam oluştur
python -m venv venv

# 2. Aktifleştir (Linux/Mac)
source venv/bin/activate

# 3. Paket kur ve listeyi kaydet
pip install requests
pip freeze > requirements.txt
```

İPUCU

Sanal ortam kullanmak, neden önemli? Çünkü A projesi bir paketin eski, B projesi yeni sürümünü isteyebilir; ikisini sistem geneline kurarsan çakışırlar. Her projeye kendi `venv`'ini vermek bu çakışmayı önler. `requirements.txt` dosyası, projenin tam paket listesini tutar; böylece başka biri (veya başka bir makine) `pip install -r requirements.txt` ile **aynı** ortamı kurabilir. `venv/` klasörünü Git'e ekleme — yalnızca `requirements.txt`'yi paylaş (Modül 9'daki Composer mantığının aynısı).

Ne yazdırır? ÇIKTI

`python -m venv venv` projeye özel izole bir ortam oluşturur; aktifleştirdikten sonra `pip install` ile kurduğun paketler yalnızca o ortama girer, sistemi etkilemez. `pip freeze > requirements.txt` ise kurulu paketleri bir dosyaya yazar — projenin "malzeme listesi" olur ve her yerde aynı kurulumu sağlar.

Alıştırma 10 dk

Ortamı düşün:

1. Sanal ortamın neden gerekli olduğunu kendi cümlenle açıkla.
2. requirements.txt dosyasının görevini yaz.
3. venv/ klasörünün neden Git'e konmadığını belirt.

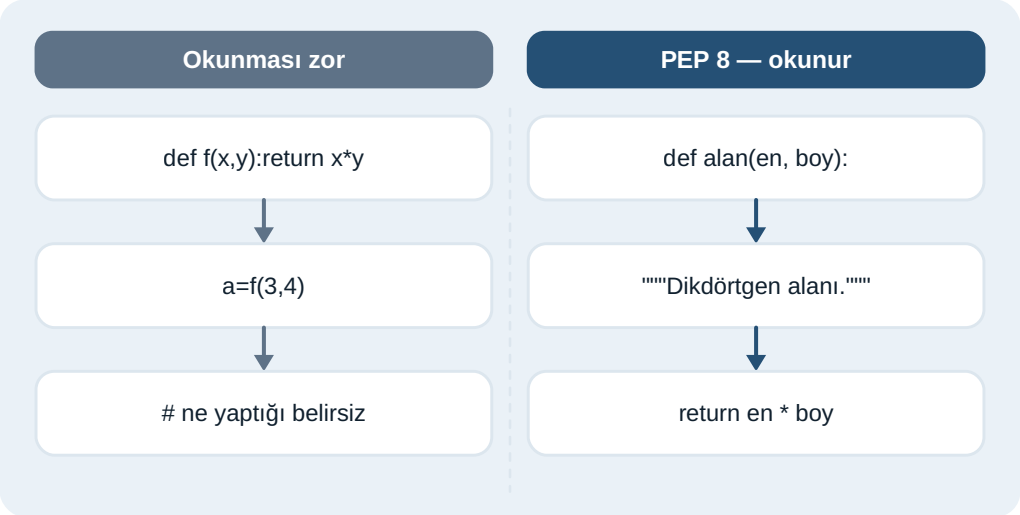
BÖLÜM 20

Kod Düzeni ve İyi Alışkanlıklar

Çalışan kod yeterli değildir; okunur kod gerekir — çünkü kodu bir kez yazar, defalarca okursun (ve başkaları da okur). Python topluluğunun ortak biçim rehberi PEP 8, herkesin kodunu tanıdık ve okunur kılar.

Okunur kod ilkeleri

- **Tutarlı girinti:** 4 boşluk (sekme değil).
- **Anlamlı isimler:** snake_case; tek harfli değişkenlerden kaçın.
- **Docstring:** fonksiyon/sınıfın ne yaptığını `"""..."""` ile açıkla.
- **Boşluk ve sadelik:** nefes alan, kısa fonksiyonlar.



Şema 20.1 — Aynı işi yapan kod: isimler ve düzen okunabilirliği belirler.

İPUCU

PEP 8, Python'un resmî stil rehberidir ve toplulukça benimsenmiştir; ona uymak, kodunu dünyadaki her Python geliştiricisi için tanıdık kılar. Pratikte tek tek kural ezberlemen gerekmez: `black` (otomatik biçimlendirici) ve `flake8` / `ruff` (denetleyici) gibi araçlar kodunu otomatik düzenler ve uyarır. **Docstring'ler** (üç tırnaklı açıklamalar) bir fonksiyonun ne yaptığını belgeler ve `help()` ile görüntülenebilir. Unutma: "Kod, yazıldığından çok daha sık okunur" — okunabilirliğe yatırım, geleceğine yatırımdır.

Ne yazdırır? ÇIKTI

İki kod parçası da aynı çarpmayı yapar; ama ikincisi anlamlı bir ad (`alan`), açık parametre isimleri (`en` , `boy`) ve bir docstring içerir — ne yaptığı bir bakışta anlaşılır. Birincisi çalışır ama okuyan kişiyi düşündürür. Okunur kod, doğru kod kadar değerlidir.

Alıştırma

10 dk

Kodu düzenle:

1. Kısaltılmış, okunması zor bir fonksiyonu PEP 8'e uygun yeniden yaz.
2. Bir fonksiyona docstring ekle.
3. "Kod yazıldığından sık okunur" sözünün ne anlama geldiğini açıkla.

SEVİYE 4

Python'u Kullanmak

Gerçek dünya: otomasyon betikleri, veriyle çalışmak, web ve API'ler, Python ve yapay zekâ, güvenlik ve sorumlu kullanım, küçük bir proje.

BÖLÜM 21

Otomasyon: Küçük Betikler

Python'un en sevilen kullanımlarından biri otomasyondur: tekrar eden, sıkıcı işleri (dosyaları yeniden adlandırmak, verileri düzenlemek, raporlar üretmek) bir betikle saniyeler içinde yaptırmak. "İki kez yapıyorsan, otomatikleştir."

Otomasyon mantığı

- Tekrar eden bir işi tanımla (örn. 100 dosyayı yeniden adlandır).
- Bir döngü + birkaç fonksiyonla betiğe dök.
- Bir kez yaz, defalarca (veya zamanlanmış) çalıştır.

Şema 21.1 — Otomasyon: saatlik manuel işi saniyelik bir betiğe çevir.

Basit bir otomasyon betiği

```
 import os

# Bir klasördeki tüm .txt dosyalarını listele
for dosya in os.listdir("belgeler"):
    if dosya.endswith(".txt"):
        print("Bulundu:", dosya)
        # burada işleyebilir, taşıyabilir, yeniden adlandırabilirsin
```

İPUCU

Otomasyonun altın kuralı: **"Bir işi ikiden fazla kez elle yapıyorsan, otomatikleştirmeyi düşün."** Python'un `os`, `shutil`, `pathlib` gibi standart modülleri dosya/klasör işlemlerini, `datetime` tarihleri kolaylaştırır. Ama dikkatli ol: dosya silen/taşıyan betikleri **önce küçük bir test klasöründe** dene — yanlış bir döngü, yüzlerce dosyayı bir anda etkileyebilir (Modül 8'deki WHERE'siz DELETE gibi). Otomasyon güçlüdür; bu yüzden önce güvenle test et.

Ne yazdırır? ÇIKTI

Bu betik "belgeler" klasöründeki dosyaları tek tek dolaşır ve `.txt` ile bitenleri bulup adını yazdırır. Buradan onları taşıyabilir, yeniden adlandırabilir veya içeriğini işleyebilirsin. Elle saatler sürecek bir iş, döngü sayesinde saniyeler içinde ve hatasız tamamlanır.

Alıştırma 12 dk

Otomasyon tasarla:

1. Günlük/haftalık tekrar eden bir işi seç ve adımlarını yaz.
2. Bir klasördeki belirli dosyaları bulan bir betik taslağı yaz.
3. Dosya değiştiren betikleri neden önce test ettiğini açıkla.

BÖLÜM 22

Veriyle Çalışmak

Python, veri işlemenin en yaygın dilidir. Bir CSV dosyasını okumak, satırları süzmek, basit özetler (toplam, ortalama) çıkarmak gibi işler, Python'da birkaç satırla yapılır. Bu, veri analizinin ilk adımıdır.

Veri işleme akışı

- **Oku:** CSV/JSON dosyasını veya API'yi yükle.
- **İşle:** süz, dönüştür, hesapla (döngü/comprehension).
- **Özetle:** toplam, ortalama, sayım; sonucu yaz.

Şema 22.1 — Veri akışı: oku → işle → özetle → sun.

Basit veri işleme

```
 import csv

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

print(f"Toplam satış: {toplam} TL")
```

İPUCU

Standart kütüphanenin `csv` modülü temel işler için yeterlidir; `csv.DictReader` her satırı bir sözlük olarak verir (başlık satırını anahtar yapar) — Modül 8'deki tablo satırlarının Python karşılığı. Daha büyük ve karmaşık veri analizleri için topluluk, güçlü dış kütüphaneler kullanır (örneğin tablolarla çalışmayı kolaylaştıran araçlar); bunlar pip ile kurulur. Veriyle çalışırken altın kural: önce veriyi **tanı** (birkaç satıra bak, tipleri kontrol et), sonra işle — kirli veya beklenmedik veri en sık hata kaynağıdır.

Ne yazdırır? ÇIKTI

Bu betik "satislar.csv" dosyasını satır satır okur, her satırın `tutar` alanını toplama ekler ve sonunda toplam satışı yazdırır. `DictReader` sayesinde her satıra sütun adıyla (`satir["tutar"]`) erişirsin. Birkaç satırlık kodla, binlerce satırlık veriden anlamlı bir özet çıkarırsın.

Alıştırma 12 dk

Veri işle:

1. Bir CSV'nin bir sütununu toplayan kod taslağı yaz.
2. Veriyi işlemeden önce neden "tanımak" gerektiğini açıkla.
3. DictReader'ın her satırı neye çevirdiğini belirt.

BÖLÜM 23

Web ve API'ler (Python ile)

Python, web'le iki şekilde çalışır: API'lerden veri çekmek (istemci) ve web uygulamaları/servisleri sunmak (sunucu). İkisi de Backend ve API modüllerinde öğrendiğin kavramların Python'daki uygulamasıdır.

API tüketmek ve sunmak

- **Tüketmek:** `requests` ile bir API'den JSON çek.
- **Sunmak:** bir web çatısı (framework) ile API/sayfa sun.
- Veri genelde JSON olarak gider gelir (Modül 7).

Şema 23.1 — Python ile API tüketmek: istek → JSON yanıt → Python verisi.

Bir API'den veri çekmek

```
import requests   # pip install requests

yanit = requests.get("https://api.ornek.com/urunler")
if yanit.status_code == 200:
    urunler = yanit.json()      # JSON -> Python listesi
    for u in urunler:
        print(u["ad"])
```

İPUCU

`requests` kütüphanesi, Python'da API tüketmenin standart yoludur. Backend modülündeki durum kodları burada işine yarar: `yanit.status_code == 200` başarıyı, 404/500 sorunu gösterir — yanıtı işlemeden önce kontrol et. API sunmak için Python'un olgun web çatıları vardır (küçük servisler ve büyük uygulamalar için ayrı seçenekler). **Güvenlik:** API anahtarlarını koda gömme (ortam değişkeni kullan, Modül 7), dış API'den gelen veriye körü körüne güvenme, ve istekleri her zaman hata yönetimiyle (try/except) sar.

Ne yazdırır? ÇIKTI

`requests.get(url)` bir API'ye istek gönderir; yanıt başarılıysa (`status_code == 200`) `.json()` dönen JSON'u bir Python listesine/sözlüğüne çevirir. Sonra bu veriyi tıpkı bir sözlük listesi gibi (Bölüm 17) işlersin. Python, web'deki veriyi programına taşımanın kolay bir yolunu sunar.

Alıştırma 12 dk

API ile çalış:

1. requests ile bir API'den veri çekme akışını adım adım yaz.
2. Yanıtı işlemeden önce neden status_code kontrol edilir, açıkla.
3. API anahtarlarının nasıl saklanması gerektiğini belirt.

BÖLÜM 24

Python ve Yapay Zekâ

Python, yapay zekâ ve makine öğrenmesinin fiilî dilidir. Veri işleme, model eğitme ve hazır YZ servislerini kullanma — hepsi Python ekosisteminde olgun araçlarla yapılır. Bu güç, beraberinde sorumluluk getirir.

Python YZ dünyasında nerede?

- **Veri hazırlama:** veriyi temizleme ve dönüştürme.
- **Model kullanma/eğitme:** hazır kütüphanelerle.
- **YZ servisleri:** hazır model API'lerini (Modül 7 gibi) çağırma.
- **Sorumlu kullanım:** her çıktıyı insan denetler.

Şema 24.1 — Python ile YZ: veriden modele — her adımda insan denetimiyle.

İPUCU

Python'un YZ'deki hâkimiyeti, zengin kütüphane ekosisteminden gelir; ama bu modülün odağı dilin kendisidir — YZ'yi derinlemesine öğrenmek ayrı bir yolculuktur (serinin 15. modülü "Kodlamada Yapay Zekâ" buna değinir). Şu ilkeleri şimdiden benimse: YZ modelleri **hata yapar ve "uydurabilir"** (halüsinasyon); çıktıları her zaman doğrula. Eğittiğin veya kullandığın veride **önyargı** olabilir — sonuçlar adil mi, sorgula. Kişisel veriyle çalışırken **KVKK ve gizlilik** kurallarına uy. Ve en önemlisi: YZ bir araçtır, **son kararı insan verir**.

Ne yazdırır? ÇIKTI

Python ile YZ akışı genelde şöyledir: veriyi topla ve temizle, bir model eğit veya hazır bir model/servis kullan, sonuçları değerlendir — ve her aşamada bir insan denetler. Python bu adımların hepsi için olgun araçlar sunar; ama aracın gücü, onu sorumlu kullanma yükümlülüğünü ortadan kaldırmaz.

Alıştırma 10 dk

Sorumlu YZ:

1. Python'un YZ'de neden bu kadar yaygın olduğunu yaz.
2. YZ çıktılarını neden her zaman doğrulamak gerektiğini açıkla.
3. Kişisel veriyle çalışırken hangi ilkelere uyulmalı, belirt.

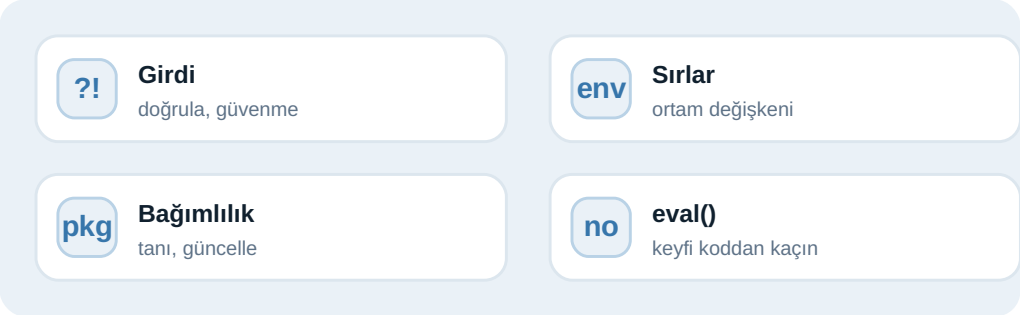
BÖLÜM 25

Güvenlik ve Sorumlu Kullanım

Python'un kolaylığı, güvenliği ihmal etme bahanesi değildir. İster bir betik ister bir web servisi yaz, birkaç temel ilke seni ve kullanıcılarını korur. Güvenlik, tüm bu seri boyunca tekrarlanan bir temadır.

Temel güvenlik ilkeleri

- **Girdiye güvenme:** kullanıcı/dış veriyi doğrula.
- **Sırları gizle:** şifre/anahtar koda değil, ortama.
- **Bağımlılık güvenliği:** paketleri tanı, güncel tut.
- **Tehlikeli işlevlerden kaçın:** `eval()` gibi keyfi kod çalıştıranlar.



Şema 25.1 — Python güvenliğinin dört temel ilkesi.

İPUCU

Bu serideki güvenlik ilkeleri dilden bağımsızdır ve Python'da da aynen geçerlidir: kullanıcı/dış veriyi **doğrula** (Modül 9), sırları **ortam değişkeninde** tut (Modül 7), veritabanı sorgularında **parametreli sorgu** kullan (Modül 8). Python'a özgü bir uyarı: `eval()` ve `exec()` gibi, bir metni kod olarak çalıştıran işlevlerden **uzak dur** — kullanıcı girdisiyle birleşince ciddi açık yaratırlar. Ayrıca dış paketlerin de bir güvenlik yüzeyi olduğunu unutma: yalnızca tanınmış, bakımı sürdürülen paketleri kur ve düzenli güncelle.

Ne yazdırır? ÇIKTI

Güvenli bir Python programı, dış veriyi doğrular, sırları koddan ayrı tutar, bağımlılıklarını bilinçli seçer ve tehlikeli işlevlerden kaçınır. Bu ilkeler küçük bir betikte de, büyük bir serviste de aynıdır. Kolay yazılan kod, güvensiz yazılmayı haklı çıkarmaz — güvenlik bir alışkanlıktır.

Alıştırma

10 dk

Güvenliği uygula:

1. Dört temel Python güvenlik ilkesini ve neye karşı koruduğunu yaz.
2. eval() gibi işlevlerin neden tehlikeli olduğunu açıkla.
3. Dış paketlerin neden bir güvenlik yüzeyi olduğunu belirt.

BÖLÜM 26

Bitirme: Küçük Bir Python Projesi

Tüm öğrendiklerini birleştirip baştan sona küçük bir proje kuruyorsun: veri al, işle, sakla ve sun. Bu, gerçek bir Python uygulamasının veya betiğinin çekirdeğidir.

Bir "harcama takipçisi" akışı

Şema 26.1 — Küçük proje: al → işle → sakla → özetle.

Proje kontrol listesi

```
 # 1. Veriyi al (girdi/dosya) ve DOĞRULA (try/except)
# 2. Fonksiyonlara böl (her biri tek iş)
# 3. Veriyi sözlük/liste ile modelle
# 4. JSON veya dosyaya KAYDET
# 5. Özet üret (toplam, ortalama) ve SUN
# 6. Sanal ortam + requirements.txt + okunur kod (PEP 8)
```

İPUCU

Gerçek bir proje yazarken bu sırayı izle: problemi **küçük fonksiyonlara böl** (Modül 6), veriyi **sözlük/listelerle modelle**, girdiyi **doğrula**, hataları **try/except** ile yakala, sonucu **dosya/JSON'a** sakla, kodu **okunur** tut ve **sanal ortamla** izole et. Bu modülü tamamladıysan, artık Python'la gerçek işler yapabilirsin: otomasyon betikleri, veri işleme, API'lere bağlanma. PHP'den sonra ikinci sunucu/genel amaçlı dilini de öğrendin — kavramların aynı, sözdizimi farklı olduğunu gördün. Sıradaki dil C# da bu deneyimin üstüne kolayca oturacak.

Ne yazdırır? ÇIKTI

Tasarladığın harcama takipçisi: kullanıcıdan/dosyadan harcamaları alır (doğrulayarak), bunları sözlük listesi olarak modeller, JSON dosyasına kaydeder ve toplam/ortalama gibi özetler üretip sunar. Veri alma, işleme, saklama ve sunma bir araya gelince — çalışan, gerçek bir Python projesi ortaya çıkar.

Alıştırma 20 dk

Projeni tasarla:

1. Küçük bir proje seç (harcama takibi, not defteri, basit hesaplayıcı).
2. Al → işle → sakla → sun akışını fonksiyonlara böl.
3. Veriyi nasıl modelleyip (sözlük/liste) saklayacağını (JSON/dosya) yaz.
4. Hangi güvenlik ve düzen ilkelerini uygulayacağını belirt.

EK

Python Terimleri ve Komutları Sözlüğü

En sık kullanılan Python yapıları ve komutları. Bir başvuru kaynağı olarak saklayabilirsin.

print()	`Ekrana yazdır`	degisken =	`Atama (\$ yok)`
f"...{x}..."	`f-string`	if / elif / else	`Koşul`
for / while	`Döngü`	range()	`Sayı aralığı`
[ ] liste	`Sıralı veri`	{ } sözlük	`Anahtar–değer`
def	`Fonksiyon`	import	`Modül/kütüphane`
with open()	`Dosya`	try / except	`Hata yönetimi`
class	`Sınıf (OOP)`	pip / venv	`Paket ve ortam`

Python'un özeti ÇIKTI

Python sade ve okunur bir dildir: **print** ile yazdırır, **değişkenlerle** (\$ yok) veri taşır, **if** ve **döngülerle** akışı yönetir — bloklar **girintiyle** belirlenir. **Listeler** ve **sözlükler** çekirdek veri yapılarıdır; **fonksiyonlar** (def) tekrar kullanılır mantığı, **sınıflar** (class) nesneleri kurar. **import** ile kütüphanelere, **with open** ile dosyalara, **try/except** ile hatalara erişirsin. Python web'den veriye, otomasyondan yapay zekâya çok geniş bir alanda kullanılır — ama güç sorumluluk getirir: girdiyi doğrula, sırları gizle, üçüncü taraf veriye güvenme.