Python Temelleri — Mosh Hamedani'den

ML Builder için Türkçe Notlar

Phase 1

2026-05-28

İçindekiler

Önsöz

0.1 Bu kitap nedir?

Bu, **Mosh Hamedani — Python for Beginners** (4 sa 27 dk YouTube kursunun) Türkçe ders notlarıdır. 12 ders halinde, Mosh'un orijinal 35 chapter'ı sistematik olarak yeniden organize edilmiş. Hedef: izlerken paralel okunabilecek; sonradan tek başına da yeterli olabilecek bir Python referansı üretmek.

Her bölüm bir **"Builder Notu"** katmanı taşır: kavramın **makine öğrenmesi** ile köprüsü. `print()` → tensor shape debug; `list` → `numpy.ndarray` → `torch.Tensor`; `dict` → `state_dict` / config; `for batch in dataloader:` → ML eğitim döngüsünün ana motoru; `class MyModel(nn.Module):` → modern AI'nın DNA'sı. Python'u "tek başına bir dil" olarak değil, ML'in lingua franca'sı olarak okuyoruz.

Kaynak

- **Video:** Mosh Hamedani — Python for Beginners (Full Course) (4 sa 27 dk, YouTube)
- **Hoca:** Mosh Hamedani — codewithmosh.com
- **Kanal:** Programming with Mosh
- **Çeviri ve genişletme:** Phase 1 (TR + ML köprüleri)

Kaynak transkript Türkçe makine-dublajıdır. Mosh quote'ları **"— Mosh (Türkçe dublaj), X:XX"** formatında verilmiştir (orijinal İngilizce değil — dublaj zaman zaman makine çevirisinin tuhaflıklarını taşır, parantez içi düzeltici notlarla işaretlenmiştir).

0.2 Nasıl Okumalı

Sıralı oku. Mosh'un her dersi bir öncekinin **kelimelerini** kullanır. Atlamak istersen, en azından *Ders 1 (Kurulum)*, *Ders 4 (Listeler/Tuple)*, *Ders 5 (Fonksiyonlar)* ve *Ders 8 (Döngüler)* zorunlu — bu dörtlü modern ML kodunun çatısını taşıyor.

Pratik bir tavsiye

Her bölüm sonundaki **egzersizleri** atlama. Özellikle PyCharm'ı açıp Mosh'la birlikte yazdığın "Guessing Game", "Translator", "Calculator" projeleri Python sezgisini parmaklarına yerleştirir. ML'de aynı kontrol akışlarını yıllarca farklı kılıklarda yazacaksın.

0.3 12 Ders

#	Ders	Mosh Ch	Ana Fikir	ML Köprüsü
1	Kurulum ve Geliştirme Ortamı	1-4	python.org + PyCharm + Hello World	Python = ML lingua franca
2	Değişkenler ve Veri Tipleri	5-7	`str`, `int`, `float`, methods	Tensor dtype, `pd.Series.str`
3	Girdi ve İlk Etkileşim	8-10	`input()`, Calculator, Mad Libs	LLM prompt template'in atası
4	Listeler ve Tuple'lar	11-13	`[...]` ve `(...)`, indexing	`ndarray` = `Tensor`
5	Fonksiyonlar	14-15, 23	`def`, `return`, parameters	`nn.Module.forward()`
6	Karar Yapıları	16-18	`if/elif/else`, `and/or/not`	`torch.where`, attention masking
7	Sözlükler ve Setler	19 + set	`{k: v}`, vocab, dedup	`state_dict`, NLP vocabulary
8	Döngüler	20-22, 24	`while`, `for`, `range`, nested	`for batch in dataloader`
9	Uygulama: Çevirmen + Yorum	25-26	string transform + `#`	NLP preprocessing, ML docs
10	Hata Yönetimi ve Dosyalar	27-29	`try/except`, `with open`	NaN guard, checkpoint persistence
11	Modüller, Paketler, Pip	30, 35	`import`, `pip install`, REPL	ML kütüphane evrenine kapı
12	Nesne Yönelimli Programlama	31-34	`class`, `__init__`, inheritance	`class MyModel(nn.Module):`

= modern AI'nın doğrudan yapı taşı.

0.4 Yazım Felsefesi

Mosh'un yaklaşımı: her kavramı küçük, çalışır örneklerle göster. Hızlı, somut, pratik. Bu kursta o yaklaşıma sadık kalıyoruz; üstüne **Builder Notu** ekliyoruz — her Python kavramının NumPy / pandas / PyTorch / scikit-learn ekosisteminde nereye düştüğü.

0.4.1 Üç Eksen

1. **Pedagojik eksen (Mosh sırası):** Mosh'un anlattığı sırayla, onun seçtiği örneklerle ilerler. Zaman damgalı Türkçe-dublaj quote'lar.
2. **Builder eksen (ML/AI bağlantıları):** Her Python kavramının kütüphane karşılığı — list comprehension → NumPy vectorization, dict → JSON config / `state_dict`, OOP → `nn.Module`, modül → kütüphane evreni.
3. **Pratik eksen (çalışır kod):** Her dersin egzersizlerinden biri runnable Python — sayıyı seçip çıktıyı kendin gör.

0.5 Builder Hedefi

Bu kurs bittiğinde, sonraki adım `numpy`, `pandas`, `torch`, `sklearn` ekosistemi. Bu kursun her dersi o adımı kolaylaştırmak için tasarlandı — Python'u ML lensinden öğrenmek.

> **!** Bir tek cümle
>
> Önceki dört kurs (lineer cebir, olasılık, calculus, derin öğrenme) sana **neyi** öğretti; bu kurs Python'la **nasıl** kodlanacağını öğretiyor — ve her ders, modern ML/AI ekosistemine açılan kapıyı işaret ediyor.

0.6 Kardeş Kurslar

Phase 1 serisi (TR + ML köprüleri):

- MIT 18.06 — Linear Algebra (Strang)
- Harvard Stat 110 — Probability (Blitzstein)
- 3Blue1Brown — Essence of Calculus
- MIT 6.S191 — Introduction to Deep Learning
- **Python Temelleri — Mosh Hamedani** (bu kitap)

1 Kurulum ve Geliştirme Ortamı

Python 3 + PyCharm + Hello World — ML yolculuğunun başlangıç noktası

Bölüm bilgisi

- **Mosh'un videosu:** Python for Beginners — Chapters 1-4 (15 dk)
- **Bölüm aralığı:** 0:00 — 15:14 (Mosh'un kursunun açılış dört chapter'ı)
- **Kaynaklar:** python.org/downloads · PyCharm Community
- **Okuma süresi:** 25 dk

1.1 Bu Derste Ne Var?

Programlamaya sıfırdan başlıyorsan, ilk yarım saatin büyük kısmı kodla değil **ortam kurmakla** geçer — ve bu, sandığından önemlidir. Yanlış sürüm Python, karışık bir IDE, bozuk bir PATH; bunlar saatlerini yer. Bu derste Mosh'la birlikte işi doğru kurup hemen çalışan bir ilk programa ulaşacağız.

Dersin dört parçası:

1. **Neden Python?** — Mosh'un girişi ve dilin "öğrenmesi neden bu kadar kolay" iddiası.
2. **Python 3'ü kurmak.** — Resmi indirme, Python 2 / Python 3 ayrımı, neden Python 3.
3. **PyCharm Community Edition.** — IDE nedir, neden gerekli, kurulum.
4. **İlk program.** — `print("Hello World")` → ASCII üçgen → sıralı yürütme kavramı.

Şekil 1.1: Ders 1'in akış haritası — kurulumdan sıralı yürütmeye

Dersin sonunda Python 3 kurulu, PyCharm açık, ilk iki programın çalışmış olacak. Daha önemlisi: Python'un küçük talimatları **sırayla** çalıştırdığını sezgisel olarak anlamış olacaksın — bu sezgi, sonraki tüm derslerin (değişkenler, döngüler, fonksiyonlar) zeminidir.

Builder Notu — Bu Dersin ML Köprüleri

- **Python = ML'in lingua franca'sı.** PyTorch, TensorFlow, scikit-learn, NumPy, pandas, JAX, Hugging Face transformers — hepsi Python. Bir ML mühendisi günde 8 saat Python yazar. Bu kurs onu **ana dil** yapma kursudur.
- **`print()` = ML'in en çok kullanılan debug aracı.** PyTorch eğitim döngüsünde `print(x.shape)`, `print(loss.item())`, `print(grad.norm())` günde yüzlerce kez çağrılır. Mosh "merhaba dünyayı bas" derken, gelecekteki tensor-shape debugging'inin temelini atıyor.
- **Sıralı yürütme = forward pass'in habercisi.** Python kodu yukarıdan aşağı satır satır çalışır — bir sinir ağının forward pass'iyle aynı yapı. `x → W x+b → ReLU → W ... → çıktı` zinciri, Mosh'un "talimatların sırası çok önemlidir" cümlesinin matematiği.
- **IDE seçimi = üretkenliğin alt yapısı.** Mosh PyCharm gösterir — gerçek dünyada ML topluluk üç IDE'ye bölünmüştür: **VS Code + Python/Jupyter extension** (en yaygın), **PyCharm Professional** (ağır refaktor), **Cursor / Claude Code** (AI-destekli). Hepsi açık. Mosh'un PyCharm Community seçimi yanlış değil — başlangıç için mantıklı.
- **Python 3 = standart.** Mosh haklı: Python 2 resmen 2020'de öldü. Hiçbir modern ML kütüphanesi Python 2 desteklemez.
- **Builder pisti.** Mosh kursun sonunda `pip install ...` görecek (Ders 11). O zaman `venv`, `conda`, `uv`, Jupyter notebook gibi modern ML iş akışları da kapıyı açacak.

1.2 Neden Python? — Mosh'un İlk Sözü

Mosh kursa doğrudan vaatle başlıyor:

> *"Bu kursta size programlamaya başlamak için bilmeniz gereken her şeyi öğreteceğim. piton."* — Mosh (Türkçe dublaj), 0:00

Türkçe dublajdaki "piton" kelimesi makine çevirisinden gelen bir tuhaflık — Mosh "Python" diyor. Öne sürdüğü dört iddia var:

1. **Python en popüler programlama dillerinden biri.** 2026 itibarıyla GitHub'da en aktif birinci-iki dilden biri (JavaScript ile rotasyon halinde). Stack Overflow'da en çok sorulan dil. TIOBE indeksinde uzun süredir zirvede.
2. **İş piyasasında en aranan dillerden.** Veri bilimi, ML/AI mühendisliği, backend, otomasyon — çoğu iş ilanında Python istenir.
3. **Güçlü ama kolay.** Mosh "cerrah dil kullanımı kolay" diyor (dublaj tuhaf, kastettiği: *succinct* — öz, kısa).

> *"temelde sadece python içinde ne yapmak istediğinizi yazmanız yeterli değil mi?"* — Mosh (Türkçe dublaj), 0:41

Java'da "Hello World" basıp çalıştırmak için:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
```

```
    }
}
```

Python'da:

```
print("Hello World")
```

Hepsi bu.

4. **Öğrenme eğrisi düşük.**

   "İlk programınızı saniyeler içinde yazmaya başlayabilirsiniz." — Mosh (Türkçe dublaj), 0:50

1.2.1 Builder Notu — ML Eki

ML işi yapmak isteyen biri için Python'ın cazibesi sadece "kolay olması" değil, **kütüphane evreni**. Mosh bunu Ders 11'de açacak (`pip`), ama bilmen gereken: scikit-learn, NumPy, pandas, PyTorch, transformers, langchain, FastAPI — hepsi `pip install` ile geliyor, hepsi Python'la yazılmış / sarmalanmış. Bir dakikanı şu cümleyi içsel olarak öğrenmeye ver: **Python'u öğrenmek = ML ekosisteminin bilet kasasına girmek.**

1.3 Python 3'ü Kurmak

Mosh'un ilk pratik adımı: Python interpreter'ını bilgisayara yükle.

1.3.1 İndirme Sayfası

python.org/downloads — resmi indirme noktası. **Hiçbir üçüncü taraf siteden indirme.**

1.3.2 Python 2 vs Python 3

Mosh tarihsel bir ayrımı açıklıyor:

> *"piton iki Pitonun eski bir versiyonudur [...] aktif olarak korunuyor ya da desteklenmiyor."* — Mosh (Türkçe dublaj), 2:57

> *"piton üç, python'un geleceği gibidir, aktif olarak korunan ve desteklenen bir sürümdür."* — Mosh (Türkçe dublaj), 3:07

Tarihsel arkaplan: Python 2.7 son sürümüydü; Python Software Foundation 1 Ocak 2020'de Python 2 desteğini resmen kesti. Mosh'un videosu Python 3.6 döneminde çekilmiş (2018-2019); 2026 itibarıyla **Python 3.12-3.13** güncel sürümler. İyi haber: Python 3.6 ile yazılan kod, Python 3.13'te hiçbir değişiklik gerektirmeden çalışır.

Tek pratik karar: En son Python 3'ü indir.

1.3.3 Platform-Spesifik Kurulum

> ⚠ Windows için kritik uyarı
>
> Windows'ta `.exe` yükleyiciyi çalıştırırken **ilk ekranda** "**Add Python 3.x to PATH**" kutusunu işaretle. Atlarsan komut satırında `python` komutu çalışmaz — saatler yer.

Platform	Komut
macOS	`.pkg` dosyası çift tıkla → "Continue" → "Install"
Windows	`.exe` çalıştır → Add to PATH → Install
Linux	`sudo apt install python3` (Debian/Ubuntu) veya `sudo dnf install python3` (Fedora)

Yükleme bittiğinde terminal açıp:

```
python3 --version
```

Çıktı `Python 3.13.x` (veya benzer) olmalı. Bu gördüğünde, Python interpreter'ı bilgisayarında çalışıyor.

1.3.4 Builder Notu — "Gerçek" ML Setup

> Builder Notu — Sanal Ortam
>
> **Gerçek ML setup'ı şudur:** sadece sistem Python'ı yetmez. Profesyonel olarak çalışan herkes **proje başına izole Python ortamı** kullanır. Seçenekler:
>
> - **`venv`** (Python'un yerleşik sanal ortam aracı). `python3 -m venv .venv` → `source .venv/bin/activate`. En basit.
> - **`conda` / `mamba`** (Anaconda ekosistemi). Veri bilimi/ML dünyasında yaygın; C kütüphane bağımlılıklarını da yönetir (CUDA, MKL).
> - **`uv`** (yeni nesil, hızlı paket yöneticisi — 2024+ standardı olmaya başladı). `uv venv` → `uv pip install ...`.
> - **`poetry`** (paket yayınlama odaklı; `pyproject.toml` + lockfile).
>
> Mosh bu derste sadece sistem Python'ını kuruyor — bu, başlangıç için doğru. Ama kursun sonuna doğru `pip install numpy` derken, projeni `.venv` içinde açmanı şiddetle tavsiye ederim. Sebep: PyTorch + CUDA + numpy + pandas + scikit-learn sürüm çatışmaları tüm sistem Python'ını kırabilir; izole ortam bu riski sıfırlar.

1.4 PyCharm Community Edition

Python interpreter'ı kurmak yeterli değil — kod yazmak için bir **editör/IDE** lazım. Mosh'un seçimi: **PyCharm Community Edition** (JetBrains).

1.4.1 IDE Nedir?

Mosh bunu kendine has tarzıyla tanıtıyor:

> *"Entegre geliştirme ortamı. temelde sadece [...] python kodumuzu çalıştırabileceğimiz özel bir ortam."* — Mosh (Türkçe dublaj), 5:20

Daha teknik tanım: **IDE (Integrated Development Environment)** = editör + interpreter çalıştırıcı + debugger + paket yöneticisi + version control entegrasyonu, tek bir uygulamada. Notepad'le de Python yazabilirsin (Mosh bunu söylüyor) ama IDE şunları yapar:

- **Otomatik tamamlama** (autocomplete) — yazarken fonksiyon/değişken önerileri.
- **Hata tespiti** — kod henüz çalışmadan önce sözdizimi hatalarını altı çizili gösterir.
- **Debugger** — kodun içine breakpoint koy, satır satır çalıştır, değişkenleri izle.
- **Yeniden adlandırma** (refactoring) — bir değişkeni 50 yerde aynı anda değiştir, güvenle.

1.4.2 PyCharm Community vs Professional

jetbrains.com/pycharm/download/ sayfasında iki sürüm:

- **Community Edition** — ücretsiz ve açık kaynak. Bu dersin tüm ihtiyaçları için yeterli.
- **Professional Edition** — ücretli (öğrenci/öğretmen olarak ücretsiz). Web framework desteği (Django, Flask), database tools, scientific tools (Jupyter, NumPy görüntüleyici).

Mosh **Community**'yi seçiyor:

> *"Topluluk sürümü var ve bu ücretsiz ve açık kaynak [...] python kullanmaya başlamak için gereken her şeye sahip olmalı."* — Mosh (Türkçe dublaj), 6:06

1.4.3 Builder Notu — IDE Pisti

PyCharm Community gerçekten doğru başlangıç. Ama ilerledikçe ML topluluğunun gerçek pratiklerini bilmek faydalı:

IDE	Kim Kullanır	Güçlü Yön
PyCharm Community	Genel Python başlangıç, web/backend	Refaktor + debugger; öğrenmesi sağlam
VS Code + Python ext	ML/AI mühendisleri (en yaygın 2026 anketleri)	Esnek, Jupyter entegre, Copilot/Continue, ücretsiz
Jupyter Notebook / Lab	Veri bilimi, ML araştırma, prototip	Hücre-hücre çalıştırma, görselleştirme, paylaşım

IDE	Kim Kullanır	Güçlü Yön
Cursor / Claude Code	AI-destekli geliştirme (yükselen)	LLM doğal entegre — kod yazma + sohbet

Pratik öneri: Bu kursta **PyCharm Community** kullan (Mosh'un yolu). Kursu bitirdikten sonra **VS Code + Python + Jupyter extension** kombinasyonuna geç — modern ML'in standardı orada. Cursor / Claude Code daha üst düzey, ileride.

1.5 İlk Program — Hello World

Mosh'un ana hedefi bu bölümde: kavramı laflarla anlatmak yerine **çalışır kod** göstermek.

1.5.1 Proje Oluştur

PyCharm açıkken **New Project**:

- **Location** — projenin klasörü. Mosh `drafts` adını veriyor; sen `python-temelleri` koyabilirsin.
- **Interpreter** — kullanılacak Python sürümü. Açılan kutudan **Python 3** seç.

Mosh önemli bir uyarı veriyor:

> *"tercüman olarak seçilen üç kişi, aksi takdirde sen Bu videoda yaptıklarımızı tam olarak takip edemeyebiliriz."* — Mosh (Türkçe dublaj), 8:16

(Dublaj kırık ama anlam: "Interpreter olarak Python 3 seçilmiş olduğundan emin ol.")

1.5.2 Yeni Python Dosyası ve İlk Satır

Sol panelde proje klasörüne sağ tıkla → **New → Python File**. Adı `app` (Mosh) ya da istediğin.

Yeni boş dosyaya tek satır yaz:

```
print("Hello World")
```

Üç parça analiz edelim:

1. `print` — Python'un yerleşik bir **fonksiyonu**. Parantez içine ne verirsen onu konsola yazdırır. Fonksiyonları Ders 5'te detaylı göreceğiz.
2. `( )` — parantezler, fonksiyonun **argümanlarını** çevreler.
3. `"Hello World"` — tırnak içinde **string** (metin dizisi). Stringleri Ders 2'de derinleştireceğiz.

1.5.3 Çalıştır

PyCharm'da üç yoldan biri:

- Sağ üstte ▶ **(Run)** düğmesi.
- Sağ tık → **Run 'app'**.
- Klavye: macOS `R`, Windows/Linux `Shift+F10`.

Konsol (alt pencere) açılır:

```
Hello World

Process finished with exit code 0
```

Tebrikler — ilk Python programını çalıştırdın. `exit code 0` = "sorunsuz bitti".

1.5.4 Builder Notu — `print()` ML'in Sırrı

`print()` ML'in en hafife alınan ama en çok kullanılan debug aracıdır. Şu satırları bir PyTorch eğitim döngüsünde günde yüzlerce kez göreceksin:

```
print(x.shape)              # tensor boyutunu kontrol et
print(loss.item())          # loss değeri
print(y_pred[:5], y[:5])    # ilk 5 tahmin vs gerçek
print(grad.norm())          # gradient'in büyüklüğü
```

Mosh "Hello World" derken aslında **tensor-shape debugging'in temelini** atıyor.

1.6 ASCII Şekil Çizimi — Birden Fazla `print`

Mosh dördüncü bölümde tek bir `print` yerine birkaç tane art arda kullanıyor ve ekrana bir **ASCII üçgeni** çiziyor.

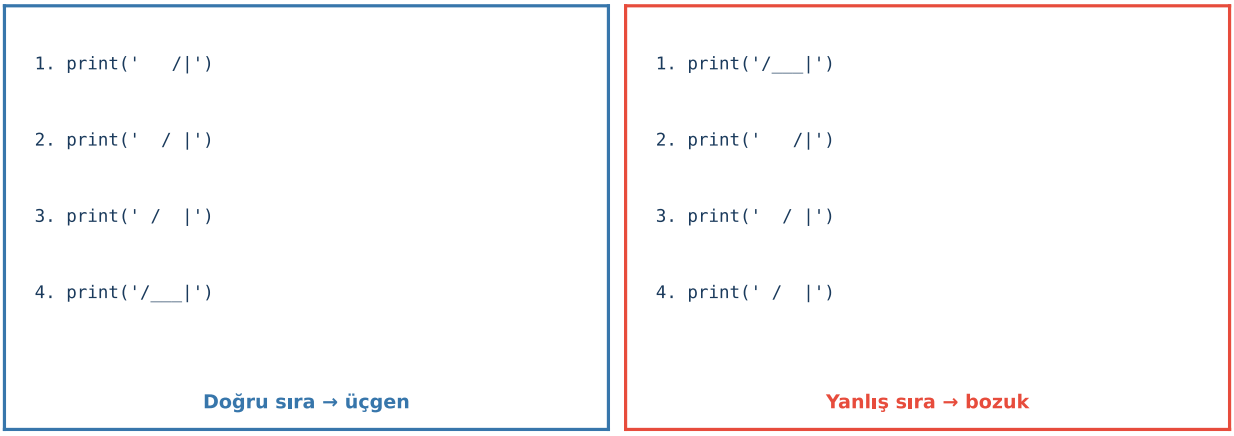
```
print("   /|")
print("  / |")
print(" /  |")
print("/___|")
```

Çalıştırınca konsolda:

```
   /|
  / |
 /  |
/___|
```

Her `print(...)` ekrana bir satır basar ve sonra otomatik bir **satır sonu** ekler.

1.6.1 Görsel — Sıralı Yürütme



Şekil 1.2: Python kodu yukarıdan aşağı satır satır çalıştırır: her print() ekrana bir satır basar. Sırayı değiştirirsen çıktı değişir.

1.7 Sıralı Yürütme — Python'un Kalbi

Bu, dersin **en önemli kavramı**.

> *"talimatların sırası aslında çok önemlidir. Bu, temel olarak python programlarınızın nasıl gideceği şeklindedir."* — Mosh (Türkçe dublaj), 14:32

1.7.1 Python = Talimat Listesi

Python **interpreter**'ı bir kod dosyasını şöyle çalıştırır:

1. Birinci satıra git → çalıştır.
2. İkinci satıra git → çalıştır.
3. Üçüncü satıra git → çalıştır.
4. … son satıra kadar.

Buna **procedural execution** (prosedürel / sıralı yürütme) denir.

1.7.2 Builder Notu — Forward Pass

Builder Notu — Sinir Ağının Kalbi

Sinir ağının forward pass'i tam olarak budur. Bir PyTorch modeli şuna benzer:

```
class Net(nn.Module):
    def forward(self, x):
        x = self.layer1(x)      # 1. talimat
        x = torch.relu(x)       # 2. talimat
        x = self.layer2(x)      # 3. talimat
        x = torch.relu(x)       # 4. talimat
        return self.output(x)   # 5. talimat
```

Python yorumlayıcısı bu satırları **yukarıdan aşağıya** çalıştırır. Her satır bir önceki satırın çıktısını alıp dönüştürür ve bir sonrakine verir. Bu zincir = forward pass.
İlginç soru: **Backward pass nasıl çalışır?** O `forward`'un *tersini* hesaplar — son satırdan başlayıp birinci satıra doğru. Bu yapı **Calculus zincir kuralı** ve **otomatik türev** (autodiff) ile sağlanır. PyTorch'un `loss.backward()` çağrısı tam olarak bu satırların ters sırada türevini alır.
Yani Mosh'un "sıra önemli" sezgisi sadece programlama değil, modern derin öğrenmenin matematik motorunun çekirdek varsayımı.

1.8 Bu Dersin Özeti

1. **Neden Python:** popüler, iş piyasasında aranan, sözdizimi sade. ML/AI'ın lingua franca'sı.
2. **Python 2 vs 3:** Python 2 (2020'de EOL), Python 3 (mevcut). Daima Python 3 öğren.
3. **Kurulum:** python.org/downloads → resmi yükleyici. Windows'ta "Add to PATH" işareti.
4. **IDE seçimi:** PyCharm Community Edition (ücretsiz, açık kaynak) bu ders için yeterli. Modern ML için VS Code + Jupyter alternatif.
5. **İlk program:** `print("Hello World")` — Python'un en sade satırı, ML eğitim döngülerinin en çok çağrılan satırı.
6. **ASCII şekil:** birden fazla `print(...)` ardı sıra → her biri bir satır basar.
7. **Sıralı yürütme:** Python kod satırlarını yukarıdan aşağıya çalıştırır. Bu, sinir ağı forward pass'inin temelidir.

! Tek Bir Cümle

Python 3'ü kur, PyCharm Community Edition'ı aç, bir `.py` dosyasına `print("Hello World")` yaz ve çalıştır — Python interpreter satırları yukarıdan aşağıya yürütür ve **sıra her şeyi belirler**; bu beş kelime sonraki tüm dersin (değişkenler, döngüler, fonksiyonlar, sinir ağları) temelidir.

1.9 Kontrol Soruları

Soru 1 — Python 2 mi, Python 3 mü?

Bir arkadaşın Python öğrenmeye başlıyor ve sana "Python 2 mi, Python 3 mü kurayım?" diye soruyor. Hangisini söyle, neden?

Cevap: Python 3 — üç katmanlı gerekçe:

1. **Resmi destek bitti.** PSF Python 2 desteğini 1 Ocak 2020'de kesti.
2. **Modern kütüphaneler Python 2 desteklemez.** PyTorch, TensorFlow, pandas, transformers — hepsi sadece Python 3.8+ destekliyor.
3. **Sözdizimi farkları küçük.** Python 3 öğrenip Python 2 koduyla karşılaşırsan birkaç saatte uyum sağlarsın.

İdeal: en son **Python 3.13** ya da kararlı 3.12.

Soru 2 — Çıktı Tahmin Et

Aşağıdaki kodu çalıştırdığında çıktı nedir?

```
print("Sabah")
print("Ogle")
print("Aksam")
```

Cevap:

```
Sabah
Ogle
Aksam
```

Her `print(...)` ekrana bir satır basar + otomatik `\n` ekler. Üç çağrı = üç satır.

Soru 3 — Mosh'un Üçgen Demosu

Mosh'un üçgen örneğinde son satırı en başa taşırsa şekil neden bozuluyor?

Cevap: Aynı dört string, farklı sıra → farklı görsel. Python kod satırlarını **yazıldığı sırada** çalıştırır. Çıktının görsel düzeni programdaki satır düzeninin doğrudan yansıması. Hiçbir Python mucizesi sırayı düzeltmez; **sıra programcının sorumluluğu**.

Soru 4 — Builder: print(loss.item())

Bir PyTorch eğitim döngüsünde `print(loss.item())` satırı ne işe yarar? Mosh'un anlattığı `print` ile aynı mı?

Cevap: Aynı `print`, aynı amaç — ölçek farklı.
Mosh `print("Hello World")` → "Hello World" basıyor. ML mühendisi `print(loss.item())` → eğitim sırasında loss değerini basıyor. Her ikisi de Python'un yerleşik `print` fonksiyonu.

```
for epoch in range(100):
    for batch in dataloader:
        loss = model(batch).loss
        loss.backward()
        optimizer.step()
        print(f"Epoch {epoch}, Loss: {loss.item():.4f}")
```

ML mühendisi günde **yüzlerce** `print` çağırır — tensor shape, gradient norm, prediction. Mosh'un birinci derste öğrettiği fonksiyon, PyTorch'la geçecek tüm gelecekte günlük araç.

1.10 Egzersizler

Egzersiz 1. Terminali aç ve `python3 --version` (macOS/Linux) veya `python --version` (Windows) komutunu çalıştır. Çıktıdaki sürüm numarasını not al. Çıktı Python 2.x veya `command not found` veriyorsa, kurulumu nereden bozulduğunu (PATH, Microsoft Store, eski yükleme) düşün ve düzelt.

Egzersiz 2. PyCharm'da yeni proje + `ders1.py` aç. Üç `print(...)` ile kendi adın, doğum yılın, bugünün tarihini bas. Çıktı dört satır mı, üç satır mı, neden?

Egzersiz 3. ASCII bir şekil çiz: kare, ok (→), kalp, ya da seçtiğin başka bir şekil. Sadece `print(...)` satırları kullan. En az 4 satırdan oluşsun. Stringlerin içindeki boşluk sayılarını dikkatle ayarla.

Egzersiz 4. (Sıralı yürütme deneyi) Şu kodu **iki kere** çalıştır — bir kere yazıldığı sırada, bir kere ortadaki iki satırı yer değiştirerek:

```
print("Adım 1")
print("Adım 2")
print("Adım 3")
print("Adım 4")
```

İki çıktı arasındaki farkı kendi sözlerinle yaz. **Bonus:** ikinci satırda kasıtlı bir hata yarat (örn. `pront("Adım 2")`) ve çalıştır. 1. satır çıkıyor mu? 3. ve 4. satırlar çıkıyor mu? Bu seni Ders 10'daki `try/except` konusuna hazırlıyor.

Egzersiz 5. (Builder eksen — sonraki dersin habercisi) Şu kodu yaz ve çalıştır:

```
ad = "Deniz"
print("Merhaba", ad)
print("Merhaba", ad, "!")
```

İki şey gözlemle:

- `print` virgülle birden fazla şey alabiliyor — bunları arada bir boşluk koyarak bastırıyor.
- `ad` bir **değişken**: bir string'i bir isimle etiketledik.

Mosh Ders 2'de tam olarak bunu (değişkenler) işleyecek. ML'de `learning_rate = 0.001`, `batch_size = 64`, `model = MyNet()` — hepsi aynı kalıp.

1.11 Sonraki Ders İçin Hazırlık

Ders 2: Değişkenler ve Veri Tipleri

Ders 2'de Mosh'un üç chapter'ına (Variables & Data Types, Strings, Numbers) gireceğiz. Bilgileri ekrana basmak iyi başlangıç ama programlar bilginin **kendisi** üzerinde çalışır — onu saklar, dönüştürür, karşılaştırır. Değişkenler bunun aracı.

- **Mosh'un Ch 5-7'sini izle** (15:14 - 48:31, ~33 dk).
- **Şu cümleyi içselleştir:** "Değişken = etiketli kutu, atama = etiket asma."

> Bu dersten tek bir şey alıp gideceksen
>
> Python kodunu yukarıdan aşağıya, satır satır çalıştırır — ve **sıra her şeyi belirler**. `print("Hello World")` Python'un en sade satırı; bir PyTorch eğitim döngüsünün en çok çağrılan satırı `print(loss.item())`. Aynı `print`, aynı zihinsel model. Mosh'un ilk yarım saatinde gördüğün küçük talimatlar, gelecek dört yılın günlük araçları.

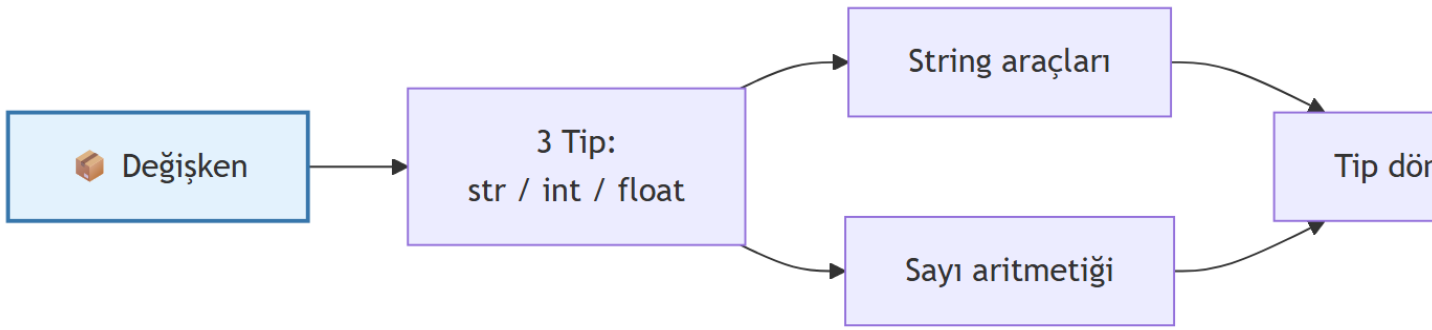
2 Değişkenler ve Veri Tipleri

`str`, `int`, `float` — Python'un ana sözlüğü

> **Bölüm bilgisi**
>
> - **Mosh'un videosu:** Chapters 5-7 (Variables → Strings → Numbers) (33 dk)
> - **Bölüm aralığı:** 15:14 — 48:31
> - **Kaynaklar:** Python docs — strings · Python docs — math module
> - **Okuma süresi:** 45 dk

2.1 Bu Derste Ne Var?

Ders 1'de `print("Hello World")` ile ekrana **sabit** metin bastık. Ama gerçek programlar sabitlerle değil, **değişen** verilerle çalışır — kullanıcının adı, bir dosyanın boyutu, bir modelin loss değeri. Bu dersin görevi: veriyi nasıl **saklarız**, **etiketleriz**, **işleriz**.



Şekil 2.1: Ders 2'nin akış haritası — değişkenden math modülüne

Dersin beş parçası:

1. **Neden değişken?** — Mosh'un hikaye motivasyonu: bir karakterin adını elle 50 yerde değiştirme derdi.
2. **Atama syntax'ı.** — `ad = "Deniz"`. Etiket-kutu zihinsel modeli.
3. **String'lerle çalışma.** — Escape (`\n`, `\"`), birleştirme (`+`), methodlar (`.upper`, `.lower`, `.replace`, `.index`), indexing (`s[0]`), `len()`.
4. **Sayılarla çalışma.** — Integer, float, dört işlem, modulus (`%`), parantezle sıra.

5. **Tip dönüşümü + `math` modülü.** — `str()`, `int()`, `float()`; `from math import *`. İlk `import`!

> Builder Notu — Bu Dersin ML Köprüleri
>
> - **Değişken = etiketli kutu = tensor referansı.** `x = torch.randn(10)` yazarsan `x` bir 10-boyutlu rastgele tensor'a etiket asar. Mosh'un `character_name = "John"`'ı zihinsel olarak aynı işlem.
> - **`snake_case` = Python standardı (PEP 8).** PyTorch'ta `learning_rate`, `batch_size`, `hidden_dim` — hepsi snake_case. `CamelCase` sadece sınıflar için (Ders 12).
> - **Dynamic typing = Python tip'i runtime'da öğrenir.** Esneklik avantaj; büyük ML kodlarında **type hints** (`x: torch.Tensor`) ile dokümente edilir (PEP 484).
> - **String indexing 0-based.** Mosh `phrase[0]` ilk karakter; numpy/PyTorch tensor indexing'iyle birebir aynı: `x[0]`.
> - **String method'ları = pandas `Series.str` ailesi.** `phrase.upper()` skalar; `df["name"].str.upper()` milyonlarca satırda. Aynı API, vektorlanmış.
> - **Aritmetik = element-wise tensor op'ları.** `3 + 4` skalar; `tensor([1,2]) + tensor([3,4]) = [4, 6]` element-wise. Aynı sembol, broadcasting ile genişletilmiş.
> - **Tip dönüşümü = ML preprocessing.** `int("42")` Mosh'un örneği; pandas'ta `df["age"].astype(int)` aynı işlem.
> - **`from math import *` = ilk kütüphane evreni.** Mosh ilk kez `import` gösteriyor. Bu satır → Ders 11'de göreceği `pip install numpy` → `import numpy as np`'ın ilkel hali.

2.2 Neden Değişken? — Mosh'un Hikaye Motivasyonu

Bu bölüm, Mosh'un en pedagojik anlatımlarından biri. Soyut bir tanımla başlamıyor; somut bir **dert** ile başlıyor: adı 50 yerde elle değiştirmek.

2.2.1 Hikaye Programı

Mosh dört satırlık bir program yazıyor:

```
print("Bir zamanlar george adinda bir adam vardi.")
print("O yetmis yasindaydi.")
print("George adini gercekten cok sevdi.")
print("Ama yetmis olmaktan hoslanmadi.")
```

Problem: **karakterin adını değiştirmek**. Mesela George'u John'a çevirelim. ne yapacağız?

> *"buraya gitmiyorum ve söyleyeceğim tamam john bunu elle değiştirmek zorunda kalacağım"*
> — Mosh (Türkçe dublaj), 16:59

Dublaj kırık ama anlam net: George'un geçtiği her satırı aç, elle değiştir. Dört satırda iki kez geçiyor — iki manuel düzeltme.

2.2.2 "Ya hikaye bin satır olsaydı?"

Mosh:

> *"bir değişken kullanabiliriz saklamak için karakterin adı ve karakterlerin yaşı. ve bu değişkeni kullandığımız zaman onu çok değiştirecek."* — Mosh (Türkçe dublaj), 18:18

Çözüm:

```
character_name = "George"
character_age = 70

print("Bir zamanlar " + character_name + " adinda bir adam vardi.")
print("O " + str(character_age) + " yasindaydi.")
print(character_name + " adini gercekten cok sevdi.")
print("Ama yetmis olmaktan hoslanmadi.")
```

Karakter adı "John" yapılacaksa:

```
character_name = "John"   # tek değiştirme
```

Tüm program otomatik güncellenir.

Builder Notu — Single Source of Truth

Bu, **tek-noktada-değişim** (single source of truth) prensibinin ilk görünüşü. ML kodunda her gün karşılaşacaksın:

```
# Kötü:
model = nn.Linear(784, 128)
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
for epoch in range(100):
    ...  # 0.001 dört yerde tekrar ediyor olsaydı...

# İyi:
LEARNING_RATE = 0.001
BATCH_SIZE = 64
HIDDEN_DIM = 128

model = nn.Linear(784, HIDDEN_DIM)
optimizer = torch.optim.SGD(model.parameters(), lr=LEARNING_RATE)
```

`LEARNING_RATE` bir kez tanımlanır; 50 satırlık bir modelde beş-on yerde kullanılır. Değiştirmek istediğinde **tek bir noktayı** değiştirirsin — Mosh'un "karakter adı"yla aynı mantık.

2.3 Atama Syntax'ı

```
ad = "Deniz"
```

Üç parça:

- **Sol taraf (`ad`)** — değişkenin adı.
- **`=`** — atama operatörü. Matematik "eşittir" değil; "ata, sakla" anlamında.
- **Sağ taraf (`"Deniz"`)** — değişkene konacak değer.

2.3.1 İsimlendirme — `snake_case` (PEP 8)

> *"farklı kelimeleri ayırmak istiyorsun bir alt çizgi ile"* — Mosh (Türkçe dublaj), 19:14

Zorunlu (Python yorumlayıcısı reddeder):

- Harf veya `_` ile başla; rakamla başlama (`5x` → SyntaxError, `x5` → OK).
- Boşluk yok (`first name` → SyntaxError, `first_name` → OK).
- Anahtar kelime kullanma (`if`, `for`, `class`, `def`, `print`, `lambda` → reserved).

Sözleşme (PEP 8):

Tür	Stil	Örnek
Değişken / Fonksiyon	`snake_case`	`character_name`, `learning_rate`
Sabit	`UPPER_SNAKE`	`MAX_EPOCHS`, `PI`
Sınıf (Ders 12)	`PascalCase`	`NeuralNetwork`, `BertModel`
Tek harf	OK (sadece kısa scope)	`i`, `n`, `x`

2.3.2 Üç Veri Tipi

```
ad = "Deniz"        # string  → tip(ad) → str
yas = 35            # integer → tip(yas) → int
boy = 1.78          # float   → tip(boy) → float
```

Python'da **tip beyan etmiyorsun** — sadece atama yap, Python tipini otomatik çıkarır (dynamic typing).

- **string (`str`)** — tırnak içinde metin. Karakterler dizisi.
- **integer (`int`)** — tam sayı. Python int sınırsız büyüyebilir.
- **float** — ondalıklı sayı. IEEE 754 64-bit hassasiyet.

Dördüncü temel tip **boolean (`bool`)** — `True` / `False`. Ders 6'da gelecek.

2.4 String'lerle Çalışma

Ch 6 (Working With Strings) Ders 2'nin en uzun bölümü (~11 dk).

2.4.1 String Yaratma

```
print("Draft Academy")          # cift tirnak
print('Draft Academy')          # tek tirnak - ayni sonuc

phrase = "Draft Academy"        # degiskene atadik
print(phrase)                   # Draft Academy
```

2.4.2 Escape Karakterleri — \n, \", \\

İçinde tırnak olan bir string nasıl yazılır? Birden fazla satıra yayılan bir metin?

```
print("Ilk satir\nIkinci satir")
```

```
Ilk satir
Ikinci satir
```

> *"buraya gelebilseydim, ve ters egik cizgi kurtarabilirim n... dizgeye yeni bir satir ekleyecegi"*
> — Mosh (Türkçe dublaj), 28:12

(Dublajda "kurtarabilirim" = "kullanabilirim". `\n` = backslash + n.)

```
print("Mosh dedi: \"piton harika\"")   # Mosh dedi: "piton harika"
print("C:\\Users\\deniz")              # C:\Users\deniz
```

> **Builder Notu — Escape'lerin ML Eki**
>
> ML'de en sık görülen escape: **path string'leri Windows yolları**.
>
> ```
> # Tehlikeli (Windows'ta yanlis okunur):
> path = "C:\Users\deniz\data\train.csv"   # \U, \d, \t = escape karmasasi
>
> # Guvenli:
> path = r"C:\Users\deniz\data\train.csv"  # r"..." = raw string, hic escape yok
> path = "C:/Users/deniz/data/train.csv"   # forward slash - tum platformlarda
> ```
>
> `r"..."` (raw string) ML kodunda regex pattern'ları ve dosya yolları için yaygın. `pathlib.Path("C:/...")` modern alternatif.

2.4.3 Birleştirme — + ve f-string

```
ad = "Deniz"
selamlama = "Merhaba " + ad + ", hos geldin!"
print(selamlama)
# Merhaba Deniz, hos geldin!
```

Modern (Python 3.6+) — f-string:

```
ad = "Deniz"
yas = 35
print(f"Merhaba {ad}, sen {yas} yasindasin!")
# Merhaba Deniz, sen 35 yasindasin!
```

Mosh f-string'i göstermez ama 2026'da yazdığın her satırda **standardın**.

2.4.4 String Methodları

```
phrase = "Draft Academy"

phrase.lower()          # 'draft academy'
phrase.upper()          # 'DRAFT ACADEMY'
phrase.isupper()        # False
phrase.upper().isupper()  # True - method chaining
phrase.replace("Draft", "Elephant")  # 'Elephant Academy'
len(phrase)             # 13
```

Builder Notu — pandas Series.str

pandas'ın `Series.str` ailesi Mosh'un öğrettiklerinin **vektorlanmış hâli**:

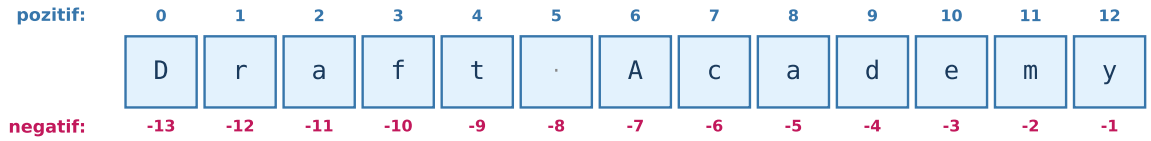
```
import pandas as pd

s = pd.Series(["Draft Academy", "Mosh Hamedani", "Python ROCKS"])

s.str.lower()                       # 3 satirin hepsi kucuk
s.str.upper().str.contains("PYTHON")  # 3 boolean
s.str.replace("Python", "Java")
s.str.len()
```

Aynı `.upper()`, aynı `.replace()`, aynı `.len` — sadece milyonlarca satırda. Mosh'un öğrettiği API **direkt** pandas'a aktarılıyor.

2.4.5 Indexing — `s[0]`, Sıfır-Tabanlı

> *"Bir dizgede indeksler kullandigimiz zaman sifir ile basliyoruz"* — Mosh (Türkçe dublaj), 34:11

```
phrase = "Draft Academy"

phrase[0]      # 'D'  (0. karakter = İLK karakter)
phrase[1]      # 'r'
phrase[-1]     # 'y'  (sondan ilk)
phrase[-2]     # 'm'
```



Şekil 2.2: Python string indexing'i — 0-tabanlı + negatif (sondan)

`s[-1]` = "son karakter" — Python'un en sevilen deyimi. ML kodunda `tensor[-1]` "son katmanın çıktısı" için sürekli.

2.4.6 Substring Arama ve Değiştirme

```
phrase.index("A")      # 6  (bosluktan sonraki A)
phrase.index("Academy") # 6  (substring baslangici)
phrase.replace("Draft", "Elephant")  # 'Elephant Academy'
```

2.5 Sayılarla Çalışma — Aritmetik

```
print(3 + 4)     # 7
print(10 - 3)    # 7
print(3 * 4)     # 12
print(10 / 3)    # 3.3333333333333335  (TRUE bolme - hep float)
print(10 // 3)   # 3                    (TABAN bolme - tam sayi)
print(10 % 3)    # 1                    (MODULUS - kalan)
print(2 ** 3)    # 8                    (KUVVET)
```

İki tür bölme:

- `/` — true division: sonuç her zaman `float`.
- `//` — floor division: aşağı yuvarlar.

2.5.1 Modulus — Tek/Çift, Periyodik

```
print(8 % 2)     # 0  (cift sayi)
print(7 % 2)     # 1  (tek sayi)
print(epoch % 10 == 0)   # her 10 epoch'ta True (ML log pattern)
```

Builder Notu — Aritmetiğin Tensor Hâli

Mosh'un öğrettiği aritmetik operatörler **tensor'da otomatik element-wise**:

```
import torch
a = torch.tensor([1, 2, 3, 4])
b = torch.tensor([10, 20, 30, 40])

a + b    # tensor([11, 22, 33, 44])
a * b    # tensor([10, 40, 90, 160])
a ** 2   # tensor([1, 4, 9, 16])
b % 3    # tensor([1, 2, 0, 1])
```

Aynı `+`, `*`, `**`, `%` sembolleri. Python'un **operator overloading** mekanizmasıyla NumPy/PyTorch/JAX bu sembolleri tensorlara genişletir.

2.6 Tip Dönüşümü

Mosh kritik bir problemi gösteriyor:

```
my_num = 5
print("Favori numaram " + my_num)   # TypeError!
# can only concatenate str (not "int") to str
```

`+` operatörü `str + int` için tanımsız. **Çözüm:**

```
my_num = 5
print("Favori numaram " + str(my_num))    # str() ile cast
print(f"Favori numaram {my_num}")          # modern: f-string
```

Diğer yön — string'i sayıya:

```
giris = "42"
sayi = int(giris)    # 42 (int)
print(float("3.14"))  # 3.14
print(int("3.14"))    # ValueError - decimal yasak
print(int(float("3.14")))  # 3 (önce float, sonra int)
```

Builder Notu — Tip Dönüşümü = ML Preprocessing

Tip dönüşümü ML pipeline'ının en yaygın işlemlerinden biri:

```
import pandas as pd
import torch

# CSV'den okuma:
df = pd.read_csv("data.csv")
df["age"] = df["age"].astype(int)        # str -> int kolon
df["score"] = df["score"].astype(float)

# Tensor tipi:
x = torch.tensor([1, 2, 3])              # int64 (default)
y = x.float()                            # float32
z = x.to(torch.float64)                  # float64
```

ML için sık karşılaşılan typecast'ler:

- **String → int** (label encoding): `"kedi"` → `0`, `"kopek"` → `1`.
- **int → float32** (NN input): tüm input'lar genelde float32.
- **uint8 → float32 / 255** (image normalization): 0-255 piksel → 0-1.

2.7 İlk `import` — `math` Modülü

Mosh ilk `import`'u tanıtıyor:

> *"ozel matematik fonksiyonlari Python matematik denilen bir sey almak zorundayim"* — Mosh (Türkçe dublaj), 46:03

```
from math import *

print(floor(3.7))    # 3
print(ceil(3.2))     # 4
print(sqrt(36))      # 6.0
print(pi)            # 3.141592653589793
print(log(100, 10))  # 2.0
```

`from math import *` neden tehlikeli?

Çakışma yaratır — örneğin `math.log` ve `numpy.log` farklı; iki `from X import *` yapan kodda hangi `log` aktif belirsiz.
Profesyonel pattern:

```python
import math
import numpy as np

math.log(100)    # 4.605... (skaler)
np.log(100)      # 4.605... (vektorize)
```

`import math as m`, `import numpy as np` — kısa aliaslarla **namespace** korunur.

2.8 Bu Dersin Özeti

1. **Değişken motivasyonu:** Mosh'un "ad-yaş" hikayesi → tek-yerde-değiştir prensibi.
2. **Atama syntax'ı:** `ad = "değer"`. `=` matematik eşittir değil.
3. **İsimlendirme:** `snake_case` (PEP 8), sabit `UPPER_SNAKE`, sınıf `PascalCase`.
4. **String araçları:** escape, concat, methodlar, `len`, `[i]`, sıfır-tabanlı.
5. **Aritmetik:** `+`, `-`, `*`, `/`, `//`, `%`, `**`. Parantezle sıra.
6. **Tip dönüşümü:** `str(5)`, `int("42")`, `float("3.14")` — ML preprocessing'in mikro hali.
7. **İlk import:** `from math import *` → ML evrenine açılan ilk kapı.

! Tek Bir Cümle

Bir değişken bir değeri bir isimle etiketler (`ad = "Deniz"`); Python üç temel tip (`str`, `int`, `float`) ile tüm temel verileri tutar; string'ler immutable ama metod-zengin (`.upper`, `.replace`, `[i]`, `len`); aritmetik operatörleri (`+ - * / // % **`) skalardan tensora kadar aynı semboller; `int()` / `str()` / `float()` tip değiştirir; ve `from math import *` Python'un kütüphane evrenine açılan ilk kapıyı aralar.

2.9 Egzersizler

Egzersiz 1. Üç değişken tanımla — ad (string), doğum yılı (int), boy metre (float). Aşağıdaki çıktıyı üret:

```
Adim Deniz, 1985 dogumlu.
Bu yil 41 yasimi dolduruyorum.
Boyum 1.78 metre, yani 178 cm.
```

Egzersiz 2. Bir string ile üç dönüşüm yap (orijinal değişmesin):

```python
mesaj = "Python ML icin essiz bir dildir"

mesaj_buyuk = ???      # Tum harfler buyuk
mesaj_js = ???         # "Python" yerine "JavaScript"
mesaj_uzunluk = ???    # Karakter sayisi
```

Egzersiz 3. Bu kodu yaz, çalıştır, çıktıları **kağıt üstünde** önceden tahmin et:

```
isim = "Hamedani"
print(isim[0])
print(isim[3])
print(isim[-1])
print(isim[-2])
print(len(isim))
print(isim[len(isim) - 1])
```

Egzersiz 4. (Aritmetik + modulus) Şu üç soruyu Python ile yanıtla:

- 100 saniye kaç dakika ve saniye? (Beklenen: `1 dakika 40 saniye`)
- 365 gün kaç hafta ve gün kalır? (Beklenen: `52 hafta 1 gun`)
- 50 sayısı çift mi? (Beklenen: `True`)

İpucu: `//` ve `%`.

Egzersiz 5. (Builder eksen) Bu kod kullanıcıdan **girdi alıyor** (Ders 3 habercisi). Eksikleri doldur:

```
ad = input("Adin: ")
yas_str = input("Yasin: ")
yas_int = ???                # yas_str'i int'e cevir
yil = 2026 - yas_int
print(f"Merhaba {ad}! {yil} yilinda dogdun.")
```

`???`'yi `int(yas_str)` ile doldur. Bu küçük örnek, Ders 3'ün ana fikrini gösteriyor: `input()` string döner; sayı işlemine girmeden önce `int()` veya `float()` ile dönüştürmek şart.

2.10 Sonraki Ders İçin Hazırlık

Ders 3: Girdi ve İlk Etkileşim

Mosh'un Ch 8-10 (Input, Calculator, Mad Libs). Program artık sadece **basmak**la yetinmeyecek — kullanıcıdan **veri almaya** başlayacak.

- **Mosh'un Ch 8-10'unu izle** (48:31 - 1:03:18, ~15 dk).
- **Şu cümleyi içselleştir:** "`input()` her zaman string döner; sayı işlemine girmeden önce `int()`/`float()` ile çevir."

> Bu dersten tek bir şey alıp gideceksen
>
> Değişken `=` ile bir değeri bir isimle etiketler; üç temel Python tipi (`str`, `int`, `float`) ve aralarındaki dönüşüm (`str()`, `int()`, `float()`) ML preprocessing'in mikro hâli; string'in metod-zenginliği (`.upper`, `.replace`, `len`, `[i]`) pandas'ın `Series.str` ailesine birebir aktarılır; ve `from math import *` ile Python'un kütüphane evrenine ilk kapıyı araladın — bu satır, sonra göreceğin `import numpy as np`'nin ilkel halinden başka bir şey değil.

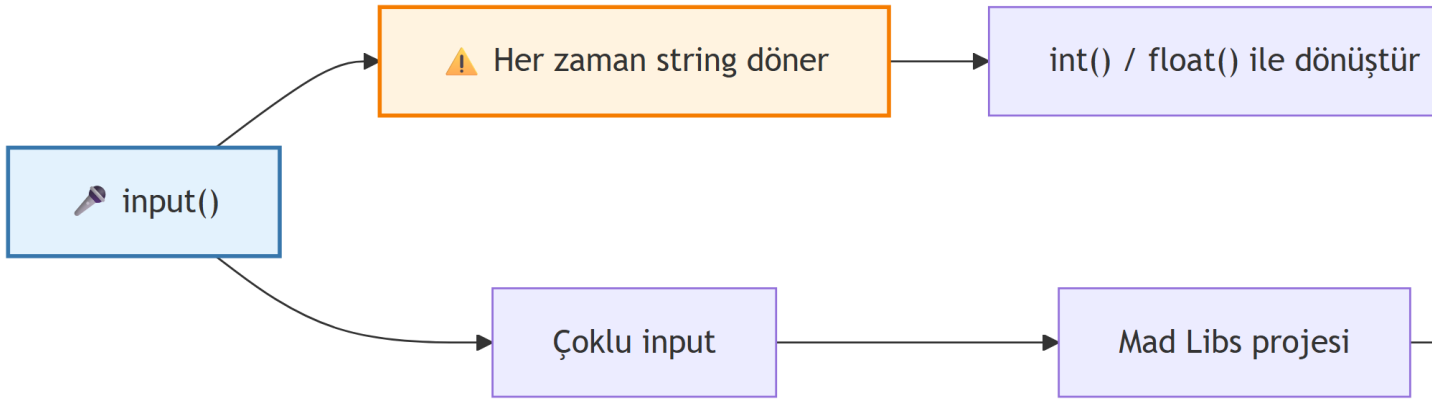
3 Girdi ve İlk Etkileşim

`input()` + Calculator + Mad Libs — LLM prompt template'in atası

> **Bölüm bilgisi**
>
> - **Mosh'un videosu:** Chapters 8-10 (Input → Calculator → Mad Libs) (15 dk)
> - **Bölüm aralığı:** 48:31 — 1:03:18
> - **Kaynaklar:** Python docs — input() · PEP 498 — f-strings
> - **Okuma süresi:** 30 dk

3.1 Bu Derste Ne Var?

Ders 1'de ekrana **basma**yı, Ders 2'de **veri saklama**yı öğrendik. Bu derste üçüncü temel etkileşim türü geliyor: kullanıcıdan **veri alma**. `input()` fonksiyonu Python'ın klavyeden okuma şekli; basit ama güçlü — tüm interaktif programların başlangıç noktası.



Şekil 3.1: Ders 3'ün akış haritası — input'tan Mad Libs'e

Dersin üç parçası (Mosh'un üç chapter'ı):

1. **`input()` fonksiyonu** — klavyeden veri çekme.
2. **Hesap makinesi (proje 1).** — **Kritik bug avı:** `input()` her zaman **string** döner.
3. **Mad Libs oyunu (proje 2).** — String birleştirmenin uygulamalı hâli.

Builder Notu — Bu Dersin ML Köprüleri

- **`input()` = ML data ingestion'ın ilkel hali.** PyTorch'ta `dataloader = DataLoader(dataset, batch_size=64)` yaparak da veri "alıyorsun" — sadece klavye yerine dosyadan/veritabanından/network'ten. Mantik ayni: bir kaynaktan veri çek, değişkende sakla, kodda kullan.
- **String-by-default = veri parse zorunluluğu.** `input()` her şeyi string döner; CSV de öyle, JSON de bazen öyle. ML pipeline'ında **parsing** (str → int/float/datetime) çıkış noktası.
- **Calculator bug = ML scaler bug analojisi.** `"5" + "8" = "58"` bug'ı, ML'de yaygın `learning_rate = "0.001"` (JSON'dan string okundu) → `optimizer.step()` kırılır tipi bug'la birebir eşlenir.
- **Mad Libs = prompt template!** Kullanıcıdan kelime al → şablon string'e yerleştir. Bu LLM çağının **prompt template'i**: `"Sen bir {role} olarak {task} yap."` LangChain `PromptTemplate`, OpenAI/Anthropic API'ların `messages` — hepsi Mosh'un mad libs'inin sofistike halleri.
- **Çoklu input = batch processing'in atası.** Mosh üç ayrı `input()` çağırısı yapıyor; ML'de bir DataLoader bir epoch'ta yüzlerce batch verisi çeker.
- **F-string'in gücü burada çıkar.** Mosh `+` ile birleştirme gösteriyor; modern Python — ve tüm ML logları — f-string kullanır: `print(f"Epoch {epoch}, loss={loss:.4f}")`.

3.2 `input()` Fonksiyonu

> *"kullanıcıdan giriş. bu yüzden temel olarak bir kullanıcının izin vermesine izin veriyoruz programimiza bilgi girişi yapın."* — Mosh (Türkçe dublaj), 48:31

3.2.1 Sentaks

```
name = input("Adin: ")
print("Merhaba " + name + "!")
```

Konsolda:

```
Adin: Deniz
Merhaba Deniz!
```

Akış:

1. Program `input(...)` satırına gelince **durur**.
2. Ekrana prompt'u (parantezdeki string) basar.
3. Kullanıcı yazıp Enter'a basar.
4. Yazılanlar **string** olarak değişkene atanır.
5. Program devam eder.

3.2.2 Çoklu Girdi

```
name = input("Adin: ")
age = input("Yasin: ")
print("Merhaba " + name + "! Sen " + age + " yasindasin.")
```

```
Adin: Deniz
Yasin: 35
Merhaba Deniz! Sen 35 yasindasin.
```

3.2.3 Kritik Detay — Her Zaman String

Bu cümle, Ders 3'ün **tek en önemli cümlesidir**:

> **!** `input()` dönüş tipi her zaman `str`'dir
>
> Kullanıcı `35` yazsa bile `age` değişkeninde saklı olan `"35"` (string), `35` (int) değil. Sayı işlemi yapmadan önce `int()` veya `float()` ile **dönüştürmek zorunlu**.

3.3 Proje 1 — Basit Hesap Makinesi (Bug Avı)

3.3.1 İlk Deneme — BUG

```
num1 = input("Birinci sayi: ")
num2 = input("Ikinci sayi: ")
result = num1 + num2
print(result)
```

Çalıştır:

```
Birinci sayi: 5
Ikinci sayi: 8
58
```

`5 + 8 = 58?` Hayır — Python toplamayı yapmadı, **stringleri birleştirdi**. `"5" + "8" = "58"` (Ders 2 §3.4: `+` string'ler için concatenation).

> *"kullanicidan girdi aldigimizda varsayilan olarak. Python sadece donusturmek icin gidiyor bir dize icine"* — Mosh (Türkçe dublaj), 54:49

3.3.2 Çözüm — `int()` veya `float()`

```
num1 = int(input("Birinci sayi: "))   # input() -> str, int(str) -> int
num2 = int(input("Ikinci sayi: "))
result = num1 + num2
print(result)
# 13
```

`int(input(...))` Python'da bir kalıp — her interaktif sayı girdisinde göreceksin.

3.3.3 İkinci Bug — Ondalıklı Sayılar

```
Birinci sayi: 4.3
Ikinci sayi: 5.5
ValueError: invalid literal for int() with base 10: '4.3'
```

`int("4.3")` hata atar — `int` ondalık nokta kabul etmez.

> *"kullanabilecegimiz baska bir fonksiyon var yerine n samandira diyebiliriz"* — Mosh (Türkçe dublaj), 57:08

(Dublajdaki "şamandıra" = `float`.)

```
num1 = float(input("Birinci sayi: "))
num2 = float(input("Ikinci sayi: "))
result = num1 + num2
print(result)
# 9.8
```

`float()` hem `"5"` (`5.0`) hem `"4.3"` (`4.3`) için çalışır. **Pratik:** bilmiyorsan `float()` kullan.

3.3.4 Tradeoff

Tip	Davranış	Ne zaman?
`int(input(...))`	Sadece tam sayı kabul; ondalık → ValueError	Tekrar sayısı, indeks, label
`float(input(...))`	Hem tam hem ondalık; sonuç hep float	Para, ölçü, ML loss değeri

3.3.5 Calculator Bug = ML Scaler Bug

⚠ Builder Notu — JSON Config Tuzağı

Mosh'un `"5" + "8" = "58"` bug'ı ML kodunda çok yaygın hâle bürünür. Senaryo: bir JSON config dosyasından hyperparameter okuyorsun.

```
{
    "learning_rate": "0.001",    // tirnak icinde - STRING!
    "batch_size": "64",
    "epochs": "100"
}
```

```
import json
with open("config.json") as f:
    cfg = json.load(f)

print(type(cfg["learning_rate"]))   # <class 'str'>  ← BUG!

# Optimizer kuralsiz reddediyor:
optimizer = torch.optim.SGD(model.parameters(), lr=cfg["learning_rate"])
# TypeError: ... must be float, not str
```

Çözüm tipi aynı Mosh'unki:

```
cfg["learning_rate"] = float(cfg["learning_rate"])
cfg["batch_size"] = int(cfg["batch_size"])
```

Veya daha güvenli — JSON'da tipleri **string olarak yazma**, doğrudan sayı:

```
{
    "learning_rate": 0.001,
    "batch_size": 64
}
```

ML mühendisinin haftada en az bir kez düştüğü tuzak.

3.4 Proje 2 — Mad Libs Oyunu

> *"deli bir libs oyun temelde sadece rastgele sozlerin bir demet girebileceginiz bir oyundur Bilirsin fiil isimleri"* — Mosh (Türkçe dublaj), 58:35

Klasik çocuk oyunu: hikaye şablonunda boşluklar var, oyuncu kelime önerir, hikaye ortaya çıkar.

3.4.1 Mosh'un Şiir Şablonu

```
Roses are red,
Violets are blue,
I love you.
```

Üç boşluk: rengi, çoğul ismi, ünlü ismi.

3.4.2 Kod

```
color = input("Bir renk soyle: ")
plural_noun = input("Bir cogul isim soyle: ")
celebrity = input("Bir unlu soyle: ")

print("Guller " + color + ",")
print(plural_noun + " mavi,")
print(celebrity + "'i seviyorum.")
```

Çalıştır:

```
Bir renk soyle: mor
Bir cogul isim soyle: mikrodalgalar
Bir unlu soyle: Tom Hanks
Guller mor,
mikrodalgalar mavi,
Tom Hanks'i seviyorum.
```

3.4.3 Modern — f-string ile

```
color = input("Bir renk soyle: ")
plural_noun = input("Bir cogul isim soyle: ")
celebrity = input("Bir unlu soyle: ")

print(f"Guller {color},")
print(f"{plural_noun} mavi,")
print(f"{celebrity}'i seviyorum.")
```

Aynı çıktı, çok daha temiz kod. **f-string Python 3.6+ standardı.**

3.4.4 Mad Libs = LLM Prompt Template

! Builder Notu — Mosh'un En Derin Habercisi

Mad Libs aslında modern LLM uygulamalarının temel mekanizmasının çocuk oyunu hâlidir: **prompt template**.

```
# Mad Libs (Mosh):
ad = input("Adin: ")
print(f"Merhaba {ad}!")

# LangChain (modern LLM):
from langchain.prompts import PromptTemplate

template = PromptTemplate(
    input_variables=["role", "task", "tone"],
    template="""
Sen bir {role} olarak yazıyorsun.
Görev: {task}
Üslup: {tone}
Yanıtın 200 kelimeyi geçmesin.
"""
)

prompt = template.format(
    role="ML mühendisi",
    task="self-attention mekanizmasını açıkla",
    tone="öğretici"
)

response = llm.invoke(prompt)
```

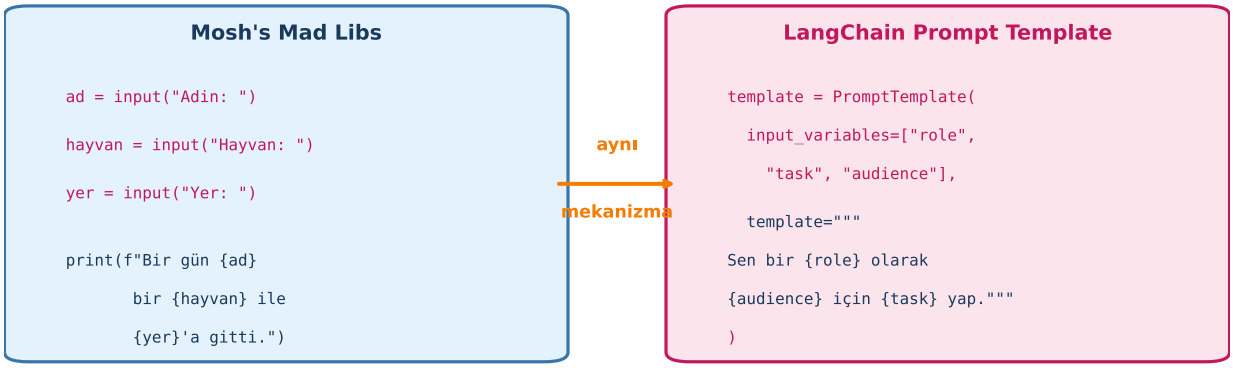
Mekanizma birebir aynı. OpenAI / Anthropic API:

```
messages = [{"role": "user", "content": f"{user_question}"}]
```

`{user_question}` f-string slot — Mad Libs'in `{ad}` slot'unun aynısı.
Mosh'un Mad Libs oyunu, LLM uygulamasının ata-formudur. Bunu içselleştirirsen, LangChain veya `openai` SDK öğrenirken "ben bunu daha önce gördüm" deyip hızla geçeceksin.

3.5 Bu Dersin Özeti

1. **`input(prompt)`** — klavyeden bir satır oku, string döndür.
2. **`input()` her zaman string döner** — kullanıcı `35` yazsa bile `"35"` saklı.
3. **Tip dönüşümü zorunlu:** `int(input(...))` veya `float(input(...))`.



Şekil 3.2: Mad Libs Prompt Template paraleli — aynı mekanizma, farklı ölçek

4. **`int` katı, `float` esnek:** `int("4.3")` ValueError, `float("4.3")` çalışır.
5. **Calculator pattern'ı:** iki sayı al → işlem → sonuç. ML scaler bug'ın ata-formu.
6. **Mad Libs pattern'ı:** çoklu input + string template. **LLM prompt template'inin atası.**
7. **f-string > `+`** — modern Python, modern ML kodu.

> **!** Tek Bir Cümle
>
> `input()` Python'ın klavyeden veri alma şeklidir ve **her zaman string döner**; sayı işlemi yapmadan önce `int(...)` veya `float(...)` ile dönüştürmek zorunludur — bu küçük dönüşüm, ML pipeline'larında her gün yaptığın preprocessing'in **mikro hâlidir**; Mad Libs ise basit string template'inin ardındaki mekanizmayla aslında LangChain / LLM prompt template'lerinin atasıdır.

3.6 Egzersizler

Egzersiz 1. Doğum yılını sor, 2026'da kaç yaşında olacağını söyle. f-string kullan.

```
Dogum yilin: 1990
2026'da 36 yasinda olacaksin.
```

Egzersiz 2. Calculator'u dört işleme genişlet (`if/elif/else` Ders 6'da; şimdilik kopyala):

```
num1 = float(input("Birinci sayi: "))
op = input("Islem (+, -, *, /): ")
num2 = float(input("Ikinci sayi: "))

if op == "+":
    print(num1 + num2)
```

```
elif op == "-":
    print(num1 - num2)
elif op == "*":
    print(num1 * num2)
elif op == "/":
    print(num1 / num2)
```

`0` ile böl, ne olur? (`ZeroDivisionError` — Ders 10 habercisi.)

Egzersiz 3. Kendi Mad Libs'ini yarat — en az **beş** input, en az **dört** satırlık şablon. f-string kullan.

Egzersiz 4. Aşağıdaki kodun çıktısını **önceden tahmin et** (girdiler: `Deniz`, `35`):

```
ad = input("Adin: ")
yas = input("Yasin: ")
print("Ad turu: " + str(type(ad)))
print("Yas turu: " + str(type(yas)))
print("Ad + Yas = " + ad + yas)
print("len(ad) = " + str(len(ad)))
print("ad[0] = " + ad[0])

print(ad + 5)    # HATA verecek - neden?
```

Son satırı düzelt.

Egzersiz 5. (Builder eksen) Mini LLM prompt builder — kullanıcıdan üç girdi al, prompt template'e yerleştir, ekrana bas:

```
role = input("Rol (ornegin: Python ogretmeni): ")
topic = input("Konu: ")
audience = input("Hedef kitle: ")

prompt = f"""Sen bir {role} olarak yaziyorsun.
Konu: {topic}
Hedef kitle: {audience}
Yanitin 200 kelimeyi gecmesin."""

print("--- GENERATED PROMPT ---")
print(prompt)
```

Bu egzersizin asıl noktası: **Mad Libs'in kendisini bir LLM prompt'a dönüştürdün**. Bu zihinsel sıçramayı yaşamak Ders 3'ün en derin kazancı.

3.7 Sonraki Ders İçin Hazırlık

Ders 4: Listeler ve Tuple'lar

Şimdiye kadar **tek** veri ile çalıştık. Ders 4'te **bir grup** veriyi tek bir yapıda saklamayı öğreneceğiz.

- **Mosh'un Ch 11-13'ünü izle** (1:03:18 - 1:24:21, ~21 dk).
- **Şu cümleyi içselleştir:** "Liste birden fazla değeri tek bir değişkende saklar — sıralı, indekslenebilir, değişebilir."

Bu dersten tek bir şey alıp gideceksen

`input()` Python'ın klavyeden veri alma şeklidir ve **her zaman string döner** — sayı işlemine girmeden önce `int()` veya `float()` ile dönüştürmek zorunlu; Mosh'un Calculator bug avı = ML'de her gün karşılaştığın "JSON'dan string okudum" bug'ın aynısı; Mad Libs'in masum hâli = modern LangChain `PromptTemplate`'in ata-formu.

4 Listeler ve Tuple'lar

`[...]` ve `(...)` — NumPy/PyTorch tensor'ların atası

Bölüm bilgisi

- **Mosh'un videosu:** Chapters 11-13 (Lists → List Functions → Tuples) (21 dk)
- **Bölüm aralığı:** 1:03:18 — 1:24:21
- **Kaynaklar:** Python docs — list · Python docs — tuple
- **Okuma süresi:** 50 dk

4.1 Bu Derste Ne Var?

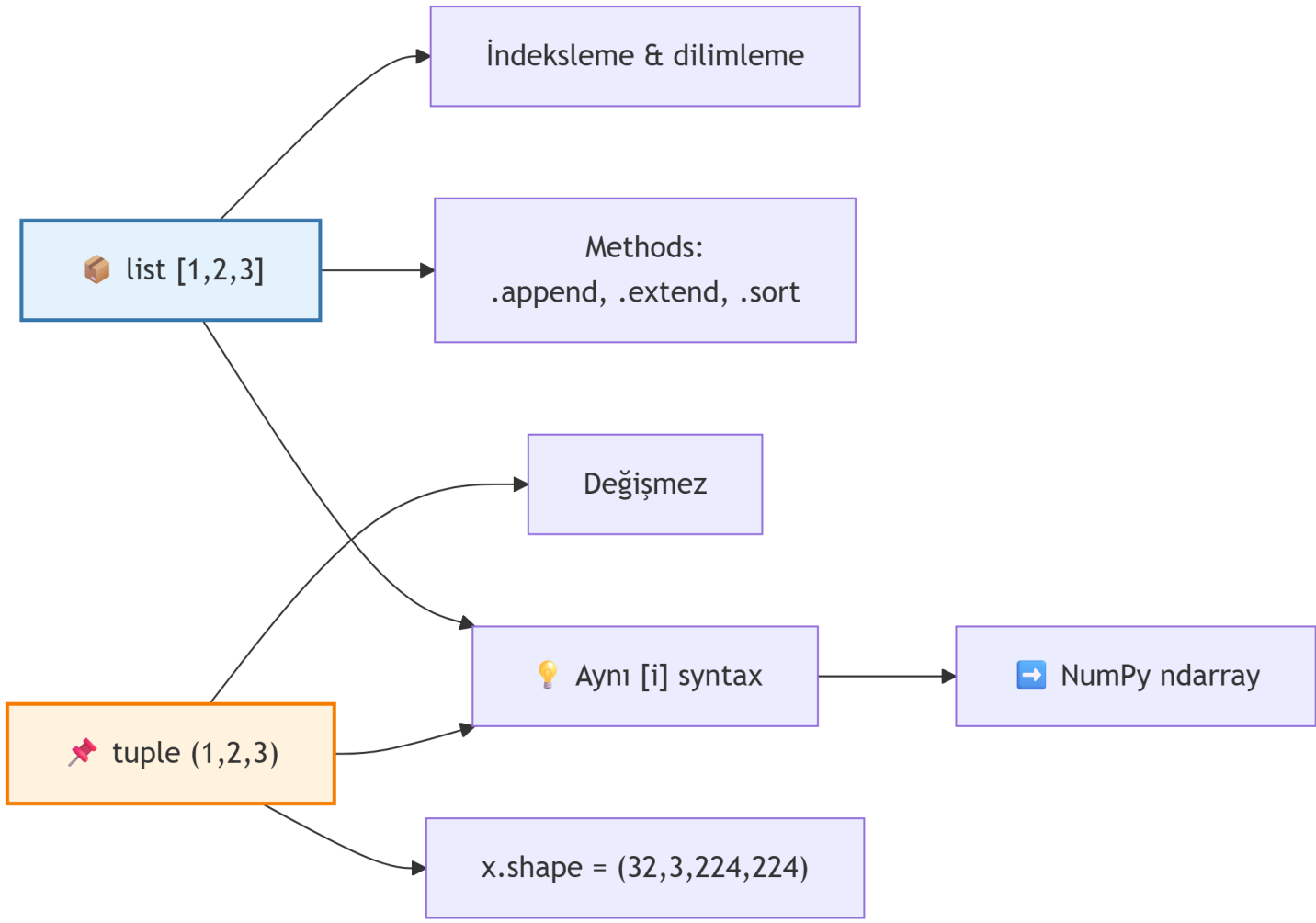
Şimdiye kadar **tek** veriyle çalıştık: bir ad, bir yaş, iki sayı. Ders 4'te **bir grup** veriyi tek bir değişkende saklamayı öğreneceksin. Bu, Python'un (ve genel programlamanın) en temel adımlarından biri — ve ML'e açılan en dolaysız köprü.

Dersin beş parçası:

1. **Listeler (`list`).** — `[1, 2, 3]`, `["Kevin", "Karen", "Jim"]`.
2. **Erişim ve dilimleme.** — `friends[0]`, `friends[-1]`, `friends[1:3]`.
3. **List metodları.** — `.append`, `.insert`, `.remove`, `.sort`, `.copy`, vb.
4. **Tuple'lar.** — `(3, 5)`. **Değişmez** (immutable).
5. **List vs Tuple.** — Ne zaman hangisi?

Builder Notu — Bu Dersin ML Köprüleri

- **List = NumPy `ndarray` = PyTorch `Tensor`'ın atası.** En derin köprü. Köşeli parantez syntax'ı tüm bu evrenlerde aynı.
- **0-tabanlı indexing = evrensel sözleşme.** Mosh'un `friends[0]`, NumPy/tensor'da `x[0]` ile aynı. `friends[-1]` "son eleman" deyimi.
- **Slicing `[a:b]` tüm Python sequence'lerinde aynı.** `x[1:5, :, ::2]` PyTorch'ta normal; Mosh'un `friends[1:3]`'ü bu syntax'ın 1-boyutlu hali.
- **`.append()` = batch oluşturma habercisi.** ML'de `losses = []` + `losses.append(loss.item())` her eğitim döngüsünde.
- **`.sort()` + `sorted()`.** ML'de top-k indeksi, scoring, ranking için her gün. `torch.sort` aynı fikir.
- **`.extend()` = `numpy.concatenate` → `torch.cat`.** Birden fazla parçayı birleştirme.



Şekil 4.1: Ders 4'ün akış haritası — list/tuple'dan tensor evrenine

- **Tuple immutable = config + shape.** Tensor shape **her zaman** tuple: `x.shape == (32, 3, 224, 224)`. Configuration veri demeti, sabit kalır.
- **Tuple unpacking = ML günlük rutini.** `batch, ch, h, w = x.shape`, `logits, hidden = model(x)`.

4.2 List Nedir, Neden Var?

> *"liste aslında sadece python içinde kullanabileceğimiz bir yapı. bilgi listelerini saklamak için bir sürü farklı veri değerlerini bir listeye koyabiliriz"* — Mosh (Türkçe dublaj), 1:03:26

4.2.1 Motivasyon

Kötü yol — tek tek değişkenler:

```
friend1 = "Kevin"
friend2 = "Karen"
friend3 = "Jim"
friend4 = "Oscar"
# ... bin tane mi yazacağız?
```

İyi yol — list:

```
friends = ["Kevin", "Karen", "Jim", "Oscar"]
```

Tek değişken, tüm değerleri kapsar. Eklemek, silmek, sıralamak — hepsi tek bir yer üzerinde.

4.2.2 Syntax

```
liste_adi = [eleman1, eleman2, eleman3, ...]
```

```
friends = ["Kevin", "Karen", "Jim", "Oscar"]    # string'ler
numbers = [42, 7, 100, -3]                       # int'ler
prices = [9.99, 24.50, 0.01]                     # float'lar
flags = [True, False, True, True]                # boolean'lar
empty = []                                       # bos liste
```

4.2.3 List, ndarray, Tensor — Akrabalar

Builder Notu — Üç Yapı, Aynı Sezgi

Üç yapı da **"sıralı dizi"**:

```
# Python list
friends = ["Kevin", "Karen", "Jim"]

# NumPy ndarray (vektorlu, ayni tip)
import numpy as np
x = np.array([1.0, 2.0, 3.0])             # shape: (3,), dtype: float64

# PyTorch Tensor (GPU + autograd)
import torch
y = torch.tensor([1.0, 2.0, 3.0])         # shape: (3,), dtype: float32

# Ayni syntax tümünde:
friends[0]   # 'Kevin'
x[0]         # 1.0
y[0]         # tensor(1.)

friends[1:3] # ['Karen', 'Jim']
x[1:3]       # array([2., 3.])
y[1:3]       # tensor([2., 3.])

len(friends) # 3
len(x)       # 3
len(y)       # 3
```

Farklar (özet):

Özellik	list	ndarray	Tensor
Türdeşlik	isteğe bağlı	zorunlu	zorunlu
Vektorlu işlem	yok	var	var
GPU	yok	yok	var
Autograd	yok	yok	var
Tipik kullanım	genel	bilimsel hesap	derin öğrenme

4.3 Erişim — Indexing ve Slicing

4.3.1 Tek Eleman

> *"bu listenin içindeki öğelerin her biri belirli bir indeks ve endeks aslında sıfırdan başlıyor"*
> — Mosh (Türkçe dublaj), 1:06:36

```
friends = ["Kevin", "Karen", "Jim", "Oscar"]

friends[0]    # 'Kevin'    (ILK = index 0)
friends[3]    # 'Oscar'    (SON = index 3)
friends[-1]   # 'Oscar'    (sondan)
friends[-2]   # 'Jim'
friends[100]  # IndexError
```

4.3.2 Slicing — `[a:b]`

```
friends[1:3]   # ['Karen', 'Jim']      (1'den 3'e, 3 HARIC)
friends[:2]    # ['Kevin', 'Karen']
friends[2:]    # ['Jim', 'Oscar']
friends[:]     # tam kopya
```

"Exclusive end" sözleşmesi: `[a:b]` index `b`'yi **dahil etmez**. Python, NumPy, PyTorch — hepsinin standardı.

4.3.3 Step ile Adım

```
nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

nums[0:10:2]   # [0, 2, 4, 6, 8]   (2'şer atla)
nums[::-1]     # [9, 8, 7, ..., 0] (TERS)
```

`nums[::-1]` Python'un meşhur "ters çevirme" deyimi.

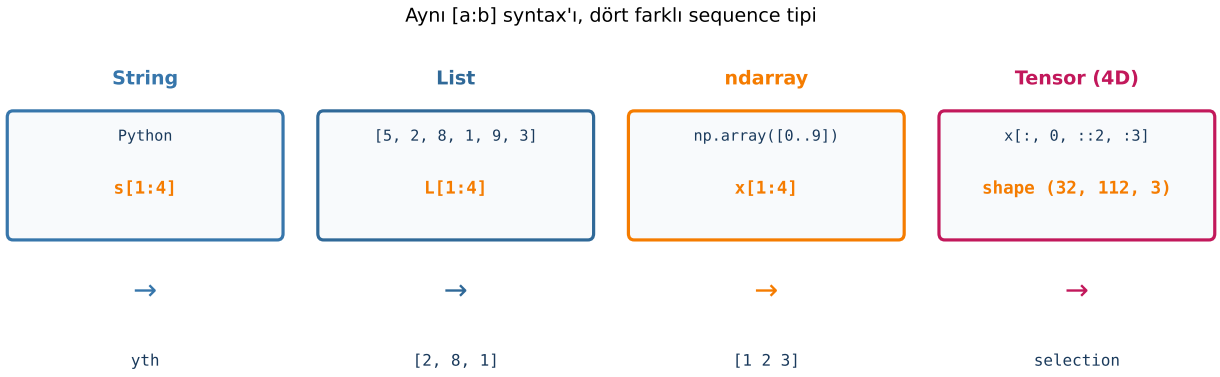
4.3.4 Slicing'in ML Eki

Builder Notu — 4-Boyutlu Slicing

NumPy ve PyTorch slicing **birebir aynı syntax**'la çalışır, ama **n-boyutlu**:

```
import torch
x = torch.randn(32, 3, 224, 224)   # batch=32, kanal=3, H=224, W=224

x[0]                  # ilk ornek: (3, 224, 224)
x[-1]                 # son ornek
x[:5]                 # ilk 5 ornek: (5, 3, 224, 224)
x[:, 0, :, :]         # her ornek - sadece R kanali: (32, 224, 224)
x[:, :, 100:150, :]   # tum orneklerin y=100-149 satirlari
x[::2, ::2, ::2, ::2] # her boyutta 2'şer atla
```

Mosh'un `friends[1:3]`'ü öğrendiğinde, yukarıdaki 4-boyutlu tensor slicing'in syntax'ı otomatik elinde. Bu Python'un en güçlü tasarım kararlarından biri.



Şekil 4.2: Slicing tüm sequence tiplerinde aynı syntax: string, list, ndarray, tensor

4.4 Mutation — List Değişebilir

String'ler immutable'dı (Ders 2): `phrase.upper()` yeni string döndürür. **List mutable**:

```
friends = ["Kevin", "Karen", "Jim", "Oscar"]
friends[1] = "Mike"
print(friends)
# ['Kevin', 'Mike', 'Jim', 'Oscar']
```

Builder Notu — Tensor Inplace Ops

ML'de `requires_grad=True` olan tensor'larda inplace mutation autograd grafiğini bozabilir:

```
w = torch.randn(10, requires_grad=True)
w[0] = 0   # UYARI: RuntimeError veya silent bug riski
```

Bu yüzden ML kodunda `x.detach()[0] = 0` veya `with torch.no_grad(): w[0] = 0` kullanılır. Mosh'un basit `friends[1] = "Mike"` mutation'ı rahat, ama tensor'larda dikkat.

4.5 List Metodları

4.5.1 `.append(x)` — Sona Ekle (EN YAYGIN)

```
friends = ["Kevin", "Karen", "Jim"]
friends.append("Oscar")
# ['Kevin', 'Karen', 'Jim', 'Oscar']
```

ML eğitim döngüsünde her epoch:

```
losses = []
for epoch in range(100):
    loss = train_one_epoch()
    losses.append(loss)
```

4.5.2 `.insert(i, x)` — Belirli Pozisyona Ekle

```
friends = ["Kevin", "Karen", "Jim"]
friends.insert(1, "Kelly")
# ['Kevin', 'Kelly', 'Karen', 'Jim']
```

4.5.3 `.extend(other)` — Listeyi Listeye Ekle

```
friends = ["Kevin", "Karen"]
new_friends = ["Jim", "Oscar"]
friends.extend(new_friends)
# ['Kevin', 'Karen', 'Jim', 'Oscar']
```

`.append` vs `.extend` farkı:

```
# Append - tek element olarak:
[1, 2].append([3, 4])    # [1, 2, [3, 4]]    iç liste!

# Extend - elemanları teker teker:
a = [1, 2]
a.extend([3, 4])         # [1, 2, 3, 4]      düz
```

4.5.4 `.remove(x)`, `.pop()`, `.clear()`

```
friends.remove("Jim")     # ilk eşleşeni sil
last = friends.pop()      # sondan çıkar ve döndür
friends.clear()           # hepsini sil
```

4.5.5 `.index(x)`, `.count(x)`

```
friends.index("Karen")    # 1 (yoksa ValueError)
[1, 2, 2, 3].count(2)     # 2
```

4.5.6 `.sort()` vs `sorted(...)`

```
nums = [42, 7, 100, -3]
nums.sort()               # YERINDE değiştirir
# nums artik [-3, 7, 42, 100]

# Non-mutation alternative:
nums2 = sorted([42, 7, 100, -3])    # yeni liste
nums2.sort(reverse=True)             # büyükten küçüğe
```

4.5.7 `.copy()` — Kopya (kritik!)

```
# YANLIS - referans paylasimi:
a = [1, 2, 3]
b = a            # AYNI liste
b.append(4)
print(a)         # [1, 2, 3, 4]  ← a da değişti!

# DOGRU:
a = [1, 2, 3]
b = a.copy()     # yeni liste
b.append(4)
print(a)         # [1, 2, 3]     ← a değişmedi
```

4.5.8 `.append()` + `.extend()` ML'de

Builder Notu — Topla, Birleştir

```
# Eğitim metrikleri:
train_losses = []
for epoch in range(EPOCHS):
    train_losses.append(train_one_epoch())

# Tensor versiyonu:
import torch
batch_logits = []
for batch in dataloader:
    batch_logits.append(model(batch))

all_logits = torch.cat(batch_logits, dim=0)   # extend gibi - tum tensor'lari birlestir
```

`torch.cat()` Mosh'un `.extend()`'inin tensor versiyonu — birden fazla tensoru tek tensora birleştirir. `numpy.concatenate()` ile aynı mantık.

4.6 Tuple — Değişmez Liste

> *"Bir tupel aslında bir listeye çok benzer [...] fakat Bir tupel listelerden bazı önemli farklılıklar vardır"* — Mosh (Türkçe dublaj), 1:19:17

4.6.1 Sentaks

```
coords = (3, 5)
print(coords)       # (3, 5)
print(coords[0])    # 3   (indexing aynı)
print(len(coords))  # 2
```

Liste `[...]`, tuple `(...)`. Tek karakter fark — büyük anlam farkı.

4.6.2 Immutability — Hata Verir

```
coords = (3, 5)
coords[0] = 10   # TypeError: 'tuple' object does not support item assignment
coords.append(7) # AttributeError: 'tuple' has no attribute 'append'
```

Tuple'da mutation **yok**. Sadece okuma: indexing, slicing, `len()`, `.index()`, `.count()`.

4.6.3 Niye Immutable?

1. **Niyet bildirimi.** "Bu veri **sabit kalacak**" demek.
2. **Hash'lenebilir → dict key olabilir.** Liste dict anahtarı olamaz; tuple olabilir (Ders 7).
3. **Performans.** Bellek olarak biraz daha verimli.

4.6.4 Tuple Unpacking — Çok Güçlü

```
nokta = (3, 5)
x, y = nokta       # x=3, y=5

# Coklu return:
ad, soyad = ("Deniz", "Demir")

# Tek satirda swap:
a, b = 1, 2
a, b = b, a        # a=2, b=1
```

C/Java/JS'de `int temp = a; a = b; b = temp;` yazmak zorundasın; Python: **tek satır**.

4.7 Tuple = ML Shape + Config

! Builder Notu — Tensor Shape ve Unpacking

Tensor shape her zaman tuple:

```
import torch
x = torch.randn(32, 3, 224, 224)
print(x.shape)         # torch.Size([32, 3, 224, 224])  ← tuple-like
print(x.shape[0])      # 32

# Unpacking - çok yaygın:
batch, channels, height, width = x.shape
```

Shape **niye tuple**? Çünkü **değişmez** — bir tensor yaratıldıktan sonra shape'i değiştirmek (`tensor.reshape(...)`) yeni tensor yaratır. Python'un "bu sabit kalacak" sinyali tam yerinde.
Model output unpacking:

```
# PyTorch transformer benzeri model:
def forward(x):
    logits = ...
    hidden_states = ...
    attention_weights = ...
    return logits, hidden_states, attention_weights   # 3-tuple

# Kullanim:
logits, hidden, attn = model(x)   # unpacking
```

dict key olarak tuple:

```
image_cache = {}
image_cache[(3, 224, 224)] = preprocessed_imagenet
image_cache[(1, 28, 28)] = preprocessed_mnist
```

ML kodunda tuple her yerde.

4.8 List vs Tuple Karar Matrisi

Soru	Cevap	Hangisi
Veri değişecek mi?	EVET	**list**
Veri sabit mi?	EVET	**tuple**
Dict key olarak?	EVET	**tuple** (zorunlu)
Birden fazla değer döndürme?	EVET	**tuple** (idiomatic)
Veri "kayıt" mi?	EVET	**tuple**
Veri "koleksiyon" mu?	EVET	**list**

```
# List - değişecek:
friends = ["Kevin", "Karen"]      # yeni arkadaş eklenebilir
losses = []                       # her epoch loss

# Tuple - sabit:
coords = (3, 5)                   # geometrik nokta
rgb = (255, 0, 0)                 # kırmızı RGB
days = ("Pzt", "Sal", "Car", "Per", "Cum", "Cmt", "Paz")
```

4.9 Bu Dersin Özeti

1. **list** — `[...]` sıralı + değişebilir.
2. **Indexing** — `friends[0]` ilk (0-tabanlı), `friends[-1]` son.
3. **Slicing** — `friends[1:3]` (3 hariç). `[::-1]` ters.

4. **Mutation** — `friends[1] = "Mike"`. List mutable, string değildi.
5. `.append(x)` — sona ekle (en yaygın).
6. `.extend(other)` — `numpy.concatenate` / `torch.cat`'in atası.
7. `.sort()` **vs** `sorted()` — yerinde vs yeni liste.
8. **tuple** — `(...)` sıralı + **değişmez**.
9. **Tuple unpacking** — `batch, ch, h, w = x.shape`.

> **!** Tek Bir Cümle
>
> Listenin köşeli parantezi (`[1,2,3]`) ile tuple'ın parantezi (`(1,2,3)`) arasındaki tek fark — **mutability** — modern ML kodunun en derin tasarım kararıdır: değişen veri liste'dir (loss history, batch buffer), sabit veri tuple'dır (tensor shape, config sabitleri); aynı `[i]` ve `[a:b]` syntax'ı string'den 4-boyutlu PyTorch tensorlarına kadar hepsinde çalışır.

4.10 Egzersizler

Egzersiz 1. En sevdiğin beş şehri liste, sonra:

```
sehirler = ["Istanbul", "Ankara", "Izmir", "Antalya", "Bursa"]

# (a) Tum listeyi yazdır
# (b) İlk ve son şehri tek satirda
# (c) Ortadaki şehir (len kullanarak)
# (d) sorted ile sırala, orijinal değişmesin
```

Egzersiz 2. Alışveriş listesi:

```
liste = []
# (a) "Ekmek", "Sut", "Yumurta" append ile
# (b) Basina "Kahve" insert ile
# (c) "Sut" remove
# (d) Yazdır → ['Kahve', 'Ekmek', 'Yumurta']
```

Egzersiz 3. Slicing tahmin et, sonra çalıştır:

```
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
print(nums[:5])
print(nums[5:])
print(nums[2:7])
print(nums[::2])
print(nums[::-1])
print(nums[1:8:2])
print(nums[-3:])
```

Egzersiz 4. İki noktanın mesafesini tuple unpacking ile:

```python
import math
p1 = (3, 4)
p2 = (7, 1)
x1, y1 = p1
x2, y2 = p2
mesafe = math.sqrt((x2-x1)**2 + (y2-y1)**2)
print(f"Mesafe: {mesafe:.2f}")    # 5.00
```

Egzersiz 5. (Builder eksen — tensor shape simülasyonu)

```python
def get_tensor_shape():
    return (32, 3, 224, 224)   # batch, channels, height, width

shape = get_tensor_shape()
batch, channels, height, width = shape

print(f"Batch boyutu: {batch}")
print(f"Kanal sayisi: {channels}")
print(f"Goruntu boyutu: {height}x{width}")
print(f"Toplam piksel: {batch * channels * height * width:,}")
# 4,816,896 piksel
```

PyTorch model çıktısı işlerken her gün bu pattern.

4.11 Sonraki Ders İçin Hazırlık

Ders 5: Fonksiyonlar

Şu ana kadar yazdığımız tüm kod düz, satır-satır akan kod. Ders 5'te kodu **yeniden kullanılabilir bloklara** ayırmayı öğreneceğiz. PyTorch'taki `nn.Module.forward(self, x)`'in dolaysız atası.

- **Mosh'un Ch 14-15'ini izle** (1:24:21-1:40:09, ~16 dk).
- **Şu cümleyi içselleştir:** "Fonksiyon = adlandırılmış kod bloğu = yeniden kullanılabilir hesaplama."

> Bu dersten tek bir şey alıp gideceksen
>
> Liste (`[...]`) değişebilir, tuple (`(...)`) değişmez — bu tek mutability farkı modern ML kodunun en derin tasarım kararıdır; aynı `[i]` ve `[a:b]` syntax'ı string'den 4-boyutlu PyTorch tensor'una kadar her yerde çalışır; Mosh'un `friends.append()`'i ML'in günlük `losses.append(loss.item())`'inin atası, `(coords)` ise PyTorch'taki `x.shape`'in geometrik akrabasıdır.

5 Fonksiyonlar

`def` + `return` — `nn.Module.forward()`'in atası

Bölüm bilgisi

- **Mosh'un videosu:** Chapters 14-15 + 23 (Functions → Return → Exponent) (18 dk)
- **Bölüm aralığı:** 1:24:21 — 1:40:09 + 2:41:25 — 2:47:16
- **Kaynaklar:** Python docs — def · PEP 8 — function names · PEP 257 — docstrings
- **Okuma süresi:** 45 dk

5.1 Bu Derste Ne Var?

Şimdiye kadar yazdığımız her kod "düzüne" çalıştı: satır 1 → satır 2 → satır 3. Yeniden kullanmak istediğimizde kopya-yapıştır ettik. Bu sağlıklı değil. Çözüm: **fonksiyon** — kodu adlandırılmış, yeniden çağırılabilir bir blokta toplama.

Şekil 5.1: Ders 5'in akış haritası — def'ten nn.Module.forward()'a

Dersin beş parçası:

1. **`def` ile fonksiyon tanımı.**
2. **Parametreler ve argümanlar.**
3. **`return` ifadesi.** — `print` ile **karıştırma!**
4. **Default + keyword argumanlar.**
5. **Scope.** — Lokal vs global.

Builder Notu — Bu Dersin ML Köprüleri

- **Fonksiyon = `nn.Module.forward()`'in atası.** Mosh'un `def greet(name): print(...)`, PyTorch'ta `class Net(nn.Module): def forward(self, x): return self.layer(x)` ile **birebir** aynı yapı. Sinir ağı = fonksiyon kompozisyonu.
- **`return` = computation graph node.** PyTorch'ta `return` ile dönen tensor autograd tarafından izlenir.

- **Parameters = hyperparameters.** `def train(lr=0.001, epochs=100):` Mosh'un parametre kavramının ML hâli.
- **Default args = kütüphane sağduyu değerleri.** `nn.Linear(bias=True)`, `nn.Dropout(p=0.5)` — kullanıcı sadece değiştirmek istediğini belirtir.
- **Pure function (saf fonksiyon) = parallelizable + cached.** JAX `jit`, PyTorch `torch.compile` — hepsi saf fonksiyon gerekir.
- **Multi-param + keyword arg = ML standartları.** `model.forward(x, mask=None, cache=None)`.

5.2 `def` ile Fonksiyon Tanımı

Mosh:

> *"fonksiyonlar bir fonksiyon sadece bir kod koleksiyonu, belirli bir görevi yerine getiren"* — Mosh (Türkçe dublaj), 1:24:21

5.2.1 Sentaks

> *"fonksiyon yazmak istersem ilk kullanmam gereken şey python içinde bir anahtar kelime* ***`def`*** *denir"* — Mosh (Türkçe dublaj), 1:25:18

```
def greet():
    print("Merhaba!")
```

Dört parça:

- `def` — anahtar kelime.
- `greet` — fonksiyon adı (`snake_case`, PEP 8).
- `()` — parametre listesi (boş).
- `:` — gövde başlangıç işareti.

Sonraki satır(lar) **girintili** olmalı — standart **4 boşluk**.

5.2.2 Çağırma

> *"Bir işlevin içindeki kod sadece onu çalıştırmak istediğimizi belirttiğimizde çalıştırılacak."* — Mosh (Türkçe dublaj), 1:28:23

Sadece `def` ile yazmak **çalıştırmaz**. Tanımlar, saklar. Çağırmak için:

```
def greet():
    print("Merhaba!")

greet()        # () ile cagri - şimdi çalışır
# Merhaba!

greet()        # bir kere daha
# Merhaba!
```

Yeniden kullanılabilirlik — fonksiyon yaratmanın en büyük kazancı.

5.2.3 Girintilik — Python'un Disiplin Kuralı

```
def greet():
    print("Merhaba!")     # girintili = fonksiyon içinde
    print("Hos geldin!")  # girintili = fonksiyon içinde

print("Bu dışarıda.")     # girinti yok
```

Java/C/JavaScript `{ }` ile belirler; Python sadece **boşluk** ile. ML kodu yüzlerce satır olabilir; sağlıklı indent disiplini olmadan okunamaz.

5.2.4 Builder Notu — PyTorch'ta `def`

Builder Notu — Modelin İskeleti

Mosh'un en basit `def greet():` örneği, PyTorch'taki model tanımının iskeletidir:

```
import torch.nn as nn

# Mosh'un seviyesi:
def greet():
    print("Merhaba!")

# PyTorch'un ayni mantigi:
class Greeter(nn.Module):
    def forward(self):
        print("Merhaba!")

# Kullanim - ayni:
greet()              # fonksiyonu çağır
model = Greeter()
model()              # PyTorch modulu cagir = forward() calisir
```

Sadece `class` sarmalı + `__call__` zekası eklenmiş. Temel kavram aynı: **fonksiyon → çağrılır**

→ **işi yapar**.

5.3 Parametreler

Sabit metin basan `greet()` çok sınırlı. Çoğu zaman **veri vermek** isteriz.

> *"fonksiyon yazdığımızda çoğu zaman, daha fazla bilgiye sahip olmak isteyeceğiz. ve bunlara parametreler denir."* — Mosh (Türkçe dublaj), 1:30:37

5.3.1 Tek Parametre

```
def greet(name):
    print("Merhaba " + name + "!")

greet("Deniz")     # Merhaba Deniz!
greet("Aylin")     # Merhaba Aylin!
greet("Mosh")      # Merhaba Mosh!
```

- **`name` parametresi** — fonksiyon **tanımında** parantezdeki isim.
- **`"Deniz"` argümanı** — fonksiyon **çağrılırken** geçilen değer.

5.3.2 Çoklu Parametre

```
def greet(name, age):
    print(f"Merhaba {name}, sen {age} yasindasin!")

greet("Deniz", 35)
greet("Aylin", 28)
```

5.3.3 Type Hints (Modern Python)

Mosh göstermez; modern ML kodunda yoğun:

```
def greet(name: str, age: int) -> None:
    print(f"Merhaba {name}, sen {age} yasindasin!")
```

Python runtime tipi **zorlamıyor** (çağrıda yanlış tip → hata olmaz) ama statik analiz (`mypy`, `pyright`) yakalar.

5.3.4 ML'in Çoklu Parametre Standardı

Builder Notu — Transformer Forward

Modern ML modelleri **çok parametreli** fonksiyonlardır:

```
class TransformerLayer(nn.Module):
    def forward(self, x, mask=None, cache=None, return_attn=False):
        # x: input tensor (batch, seq_len, d_model)
        # mask: opsiyonel attention mask
        # cache: opsiyonel KV cache (generation icin)
        # return_attn: attention weights donsun mu?
        ...
        return output
```

Dört parametre: `x` (zorunlu), `mask`, `cache`, `return_attn` (üçü default'lu). Mosh'un `def greet(name, age):` öğrettiği yapı, bu modelin forward fonksiyonunun çekirdek pattern'ı. PyTorch konvansiyonları:

- **İlk: `self`** — instance referansı (Ders 12).
- **İkinci: `x`** — ana tensor input.
- **Sonraki: `mask`, `cache`, vs.** — opsiyonel modifier'lar, default'lu.

5.4 `return` İfadesi

> *"bazen bir fonksiyon çağırdığımızda aslında almak isteyeceğiz bilgi geri Bu fonksiyondan."*
> — Mosh (Türkçe dublaj), 1:34:40

5.4.1 `print` vs `return` — KRİTİK AYRIM

İki kavram çok karıştırılır:

```
def hesapla_kup_print(num):
    print(num * num * num)    # ekrana basar, deger DONMEZ

def hesapla_kup_return(num):
    return num * num * num    # deger DONER, ekrana basmaz
```

Karşılaştır:

```
# print versiyon:
x = hesapla_kup_print(3)
# Ekrana: 27
print(x)
```

```
# Ekrana: None        ← fonksiyon hicbir sey donmedi!

# return versiyon:
y = hesapla_kup_return(3)
# Ekrana: (hicbir sey)
print(y)
# Ekrana: 27          ← y artik 27, kullanilabilir
```

! ML'de Kritik

Model fonksiyonları **her zaman** `return` etmelidir. Yaygın yeni-başlayan bug:

```
# YANLIS:
def model_forward(x):
    output = self.layer(x)
    print(output)        # ekrana yazar ama dondurmez

loss = model_forward(x)
loss.backward()          # AttributeError: 'NoneType' object has no attribute 'backward'
```

Pratik kural: Print debug için, return değer iletişimi için.

5.4.2 Birden Fazla Değer

Ders 4'te (Tuple Unpacking) öngörmüştük:

```
def istatistik(nums):
    return min(nums), max(nums), sum(nums) / len(nums)

en_kucuk, en_buyuk, ortalama = istatistik([5, 2, 8, 1, 9, 3])
```

`return a, b, c` aslında `return (a, b, c)` — tuple döner. PyTorch'ta:

```
output, hidden = lstm_layer(x)            # 2 deger
logits, attn = transformer(x, return_attn=True)
```

5.4.3 Erken `return` ile Guard Clause

```
def bol(a, b):
    if b == 0:
        return None       # erken cikis - hata durumunda None
    return a / b
```

```
print(bol(10, 2))       # 5.0
print(bol(10, 0))       # None
```

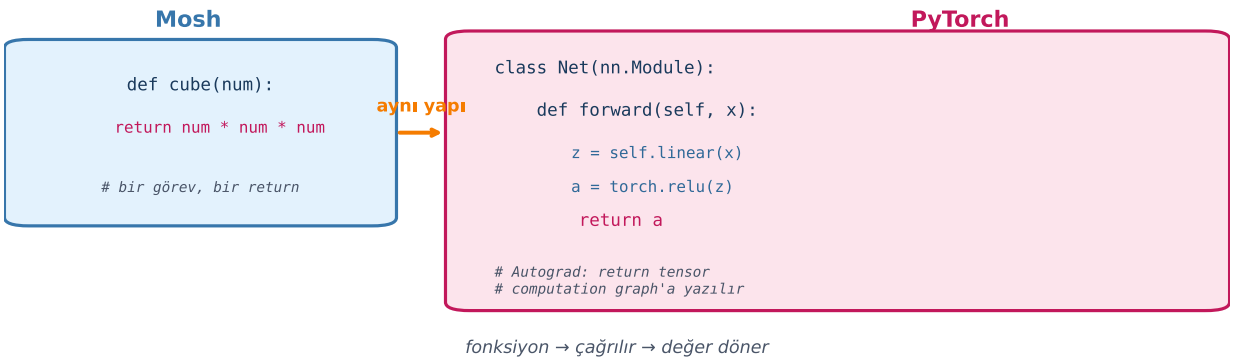
5.4.4 `return` = Computation Graph Node

Builder Notu — Autograd Zinciri

PyTorch'ta `return` ile döndürülen tensor **autograd graph**'ında bir düğümdür:

```
import torch

def my_model(x, w, b):
    z = x @ w + b               # linear transform
    a = torch.relu(z)           # activation
    return a                    # bu DUGUM autograd ile izlenir

w = torch.randn(3, 5, requires_grad=True)
b = torch.randn(5, requires_grad=True)
x = torch.randn(10, 3)

output = my_model(x, w, b)
loss = output.sum()
loss.backward()                  # autograd, my_model'den geri gider
print(w.grad)                    # w'nin gradient'i hesaplanmis
```

PyTorch `loss.backward()` çağrısında bu satırların türevini alır. Mosh'un mikro `return num * num * num`'ı, ML autograd sisteminin atasıdır.



Şekil 5.2: Mosh'un `def cube(num): return num**3` mikro örneği vs PyTorch forward — aynı yapı

5.5 Default Argumanlar

Mosh göstermez ama modern Python'da **her yerde**.

```
def greet(name="Misafir"):
    print(f"Merhaba {name}!")

greet()              # Merhaba Misafir!     (argüman yok - default)
greet("Deniz")       # Merhaba Deniz!       (argüman default'u eziyor)
```

5.5.1 Birden Fazla Default + Keyword Argument

```
def kullanici_yarat(ad, yas=18, sehir="Istanbul", rol="user"):
    print(f"{ad} ({yas}), {sehir}, rol: {rol}")

# Sirayla:
kullanici_yarat("Deniz")
# Deniz (18), Istanbul, rol: user

# Karısık - keyword ile:
kullanici_yarat("Deniz", rol="admin")
# Deniz (18), Istanbul, rol: admin

kullanici_yarat(ad="Aylin", sehir="Ankara", yas=28)
# Aylin (28), Ankara, rol: user
```

`name=value` syntax'ı **isim ile** belirtir. Sıra önemsiz.

⚠ Mutable Default Tuzağı

Yaygın Python tuzağı:

```
# YANLIS:
def liste_ekle(yeni, liste=[]):
    liste.append(yeni)
    return liste

print(liste_ekle(1))    # [1]
print(liste_ekle(2))    # [1, 2]  ← HATA! Onceki cagrinin sonucu kaldi
print(liste_ekle(3))    # [1, 2, 3]
```

Default `[]` fonksiyon tanımlandığında **bir kere** yaratılır; tüm çağrılar **aynı** listeyi paylaşır.
Doğru pattern:

```python
def liste_ekle(yeni, liste=None):
    if liste is None:
        liste = []
    liste.append(yeni)
    return liste
```

Mutable default (list, dict, set) yerine `None` kullan ve fonksiyon içinde initialize et.

5.5.2 ML Kütüphanelerinde Default'lar

PyTorch katmanları çok parametreli + default'lu:

```python
import torch.nn as nn

# nn.Linear: bias=True default
layer = nn.Linear(in_features=784, out_features=128)   # bias eklenir
layer_no_bias = nn.Linear(784, 128, bias=False)

# nn.Conv2d: bir sürü default
conv = nn.Conv2d(
    in_channels=3,
    out_channels=64,
    kernel_size=3,
    stride=1,           # default
    padding=0,          # default
    dilation=1,         # default
    bias=True,          # default
)
```

Eğitim fonksiyonu tipik hali:

```python
def train(
    model,
    train_loader,
    val_loader,
    epochs=100,
    lr=1e-3,
    weight_decay=1e-5,
    log_interval=10,
    device="cuda",
    early_stopping=True,
    patience=5,
):
    ...
```

On parametre, dokuzu default.

5.6 Scope — Lokal ve Global

Fonksiyon **kendi içinde** değişkenler yaratabilir. Bu değişkenler **dışarıdan görünmez**.

5.6.1 Lokal Değişken

```
def hesapla():
    x = 10
    y = 20
    return x + y

print(hesapla())     # 30
print(x)             # NameError: name 'x' is not defined
```

5.6.2 Global Değişken

```
LEARNING_RATE = 0.001     # global

def train():
    print(f"LR: {LEARNING_RATE}")     # global'i okur

train()                   # LR: 0.001
```

Ama **yazmak** için `global` keyword:

```
counter = 0

def increment():
    global counter
    counter += 1
```

5.6.3 Pure Function — Yan Etkisi Olmayan

ML'in en sevdiği pattern:

```
# Pure - iyi:
def kare(x):
    return x * x

# Kirli (global modification):
toplam = 0
def ekle(x):
    global toplam
    toplam += x
```

Builder Notu — Saf Fonksiyon = ML Performansı

JAX'in `jit` decorator'ı saf fonksiyon zorunluluğu:

```
import jax

@jax.jit
def model_forward(x, w, b):
    return jax.nn.relu(jax.numpy.dot(x, w) + b)
```

Bu fonksiyon:

- Sadece `(x, w, b)`'ye bağlı.
- Yan etkisi yok.
- Çıktı deterministik.
- Paralel çalıştırılabilir.
- GPU/TPU'da JIT compile edilir.

PyTorch'ta `torch.compile`, modern Python'da `functools.cache` — hepsi saf fonksiyon gerekir. Mosh'un öğrettiği scope kavramı, modern ML pratiğinin felsefi temeli.

5.7 Docstring — Fonksiyonu Belgele

```
def kup(num):
    """Verilen sayinin kupunu doner.

    Args:
        num: Kupü alinacak sayi (int veya float).

    Returns:
        num * num * num
    """
    return num * num * num
```

Üç tırnak (`"""..."""`) ile yazılan ilk string = docstring. `help(kup)` ile görünür. PyTorch ve NumPy'nin milyonlarca satırlık kodu docstring'lerle dolu.

5.8 Bu Dersin Özeti

1. **`def name(params):`** — fonksiyon tanımı.
2. **Çağrı `name(args)`** — fonksiyonu çalıştırır.
3. **`return x`** — değer döndürür. Yoksa `None`.
4. **`return  print`** — KARIŞTIRMA. ML'de kritik.

5. **Default arg `def f(x=0):`** — opsiyonel parametre.
6. **Keyword arg `f(x=5)`** — sırayı önemsizleştirir.
7. **Lokal scope** — fonksiyon içi değişkenler dışarıdan görünmez.
8. **Pure function** — yan etkisi yok = ML altın standart.
9. **Mutable default tuzağı** — `def f(x=[]):` YASAK; `None` sentinel.
10. **Docstring `"""..."""`** — belgele.

> **!** Tek Bir Cümle
>
> Fonksiyon = adlandırılmış, parametre alabilen, değer döndürebilen kod bloğudur (`def name(params): return value`) — Python'da gerçek programcı olmanın eşiği ve PyTorch'taki `nn.Module.forward(self, x)`'in dolaysız atasıdır; `return` ile döndürülen tensor autograd grafiğinde bir düğüm olur ve `loss.backward()` ile takip edilir; default argümanlar modern ML kütüphanelerinin standardıdır; lokal scope disiplini = saf fonksiyon disiplini = JAX `jit`, PyTorch `torch.compile` performans optimizasyonlarının temelidir.

5.9 Egzersizler

Egzersiz 1. Üç matematik fonksiyon — kare, küp, ondördüncü kuvvet:

```
def kare(x):
    return ???

def kup(x):
    return ???

def ondord_kuvvet(x):
    return ???

print(kare(5))           # 25
print(kup(3))            # 27
print(ondord_kuvvet(2))  # 16384
```

Egzersiz 2. Selamlama — `name` zorunlu, `selam`+`noktalama` default:

```
def selamla(name, selam="Merhaba", noktalama="!"):
    print(f"{selam} {name}{noktalama}")

# 5 çağrı için çıktıyı önceden tahmin et:
selamla("Deniz")
selamla("Aylin", "Selam")
selamla("Mosh", noktalama=".")
selamla(name="Steve", selam="Hi")
selamla("Kelly", noktalama="?", selam="Naber")
```

Egzersiz 3. (Multi-return) Liste istatistiği:

```
def istatistik(nums):
    return ???   # min, max, mean, sum tuple

nums = [5, 2, 8, 1, 9, 3, 7, 4, 6]
en_kucuk, en_buyuk, ortalama, toplam = istatistik(nums)
```

Egzersiz 4. (Saf fonksiyon disiplini) Aşağıdaki yan-etkili kodu **saf** hale getir:

```
# YAN ETKİLİ:
toplam_skor = 0
notlari = []

def not_ekle(yeni):
    global toplam_skor
    notlari.append(yeni)
    toplam_skor += yeni
    print(f"Yeni: {yeni}, toplam: {toplam_skor}")
```

Saf versiyonu yaz — global yok, print dışında.

Egzersiz 5. (Builder eksen — fake neural network forward)

```
def linear_layer(x, w, b):
    return ???   # y = x * w + b

def relu(x):
    return ???   # max(0, x)

def model_forward(x, w1, b1, w2, b2):
    z1 = linear_layer(x, w1, b1)
    a1 = relu(z1)
    z2 = linear_layer(a1, w2, b2)
    return z2

x = 3.0
w1, b1 = 2.0, 1.0
w2, b2 = 0.5, -1.0

output = model_forward(x, w1, b1, w2, b2)
print(f"Output: {output}")
# Beklenen: 2.5 (3*2+1=7, relu(7)=7, 7*0.5-1=2.5)
```

Mosh'un öğrettiği fonksiyon kavramının **gerçek PyTorch model forward pattern'ı** ile birebir aynı yapı olduğunu gösterir. Sadece skaler yerine tensor, ve `requires_grad=True` ile autograd çalışır.

5.10 Sonraki Ders İçin Hazırlık

Ders 6: Karar Yapıları

Mosh'un Ch 16-18 (If, Comparisons, Better Calculator). Şu ana kadar kod doğrusal akıyordu; Ders 6'da **dallandırma** ekleyeceğiz.

- **Mosh'un Ch 16-18'ini izle** (1:40:09-2:07:20, ~27 dk).
- **Şu cümleyi içselleştir:** "`if` programa dal seçme yeteneği verir."

> Bu dersten tek bir şey alıp gideceksen
>
> `def name(params): return value` — adlandırılmış kod bloğu, parametrelerle veri al, `return` ile değer ver — modern Python'un ve PyTorch'un (`class Net(nn.Module): def forward(self, x): return ...`) iskeletidir; saf fonksiyon disiplini JAX `jit` ve PyTorch `torch.compile` modern performans araçlarının zeminidir; Mosh'un üç chapter'ında öğrettiği basit `def kup(num): return num*num*num`, ML eğitim döngüsündeki her `def forward(self, x): return self.layer(x)`'in çekirdek atasıdır.

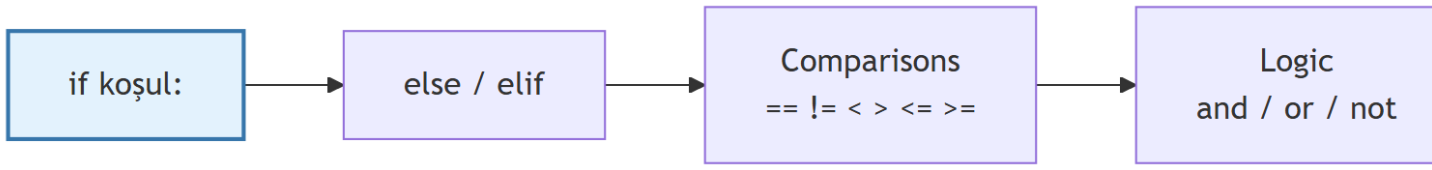
6 Karar Yapıları

`if/elif/else` + comparison + logic — `torch.where` ve attention masking'in atası

> **Bölüm bilgisi**
>
> - **Mosh'un videosu:** Chapters 16-18 (If → Comparisons → Better Calculator) (27 dk)
> - **Bölüm aralığı:** 1:40:09 — 2:07:20
> - **Kaynaklar:** Python docs — control flow · PEP 308 — conditional expressions
> - **Okuma süresi:** 50 dk

6.1 Bu Derste Ne Var?

Şimdiye kadar kod hep yukarıdan aşağı **doğrusal** çalıştı. Ders 6'da ilk **dallanmayı** ekleyeceğiz: "şu durumda bunu yap, başka durumda diğerini." Bu, Python'u (ve genel olarak programlamayı) "düz hesaplayıcı"dan **karar verebilen** akıllı bir araca dönüştürür.



Şekil 6.1: Ders 6'nın akış haritası — if'ten torch.where'e

Dersin beş parçası:

1. **`if` ifadesi.** — `if hungry: eat()`.
2. **`else` ve `elif`.** — Çoklu dal.
3. **Karşılaştırma operatörleri.** — `==`, `!=`, `<`, `>`, `<=`, `>=`.
4. **Mantık operatörleri.** — `and`, `or`, `not`.
5. **Better Calculator (proje).** — `if/elif/else` ile dispatch.

Builder Notu — Bu Dersin ML Köprüleri

- **`if/else` = `torch.where()` masking.** Mosh'un `if x > 0: y = x else: y = 0` (ReLU mantığı!), PyTorch'ta `y = torch.where(x > 0, x, torch.zeros_like(x))` ile birebir aynı iş. **Element-wise** çalıştığı için batch verisinde yüz binlerce karar tek hamlede.
- **Karşılaştırma = boolean mask.** `x > 0` Python'da skaler True/False; PyTorch'ta `tensor > 0` → **tensor of True/False** (boolean mask). Bu mask ile filtreleme: `x[x > 0]` (pozitif elemanlar).
- **Boolean = mask tensor.** ML'de "hangi token padding, hangi değil", "hangi pixel arka plan" — hepsi `True/False` tensor'lar.
- **`and/or/not` = `&`, `|`, `~` tensor'larda.** Python'da `True and False` boolean; PyTorch'ta `mask1 & mask2` (element-wise AND). Multi-condition filtering ML'de kritik.
- **Multi-branch (`elif`) = `argmax` / `bucketize`.** `if score > 0.9: A elif score > 0.5: B else: C` mantığı, softmax + threshold ile vektörlü.
- **Truthy/Falsy = empty check pattern.** `if liste:` boş mu kontrolü Python'un en sevilen deyimi.
- **Short-circuit = lazy evaluation.** `if obj is not None and obj.value > 0:` — güvenli null kontrol.
- **Sanity check disiplini.** ML'de `if loss.isnan(): raise RuntimeError(...)` — üretim ML kodunun savunma mekanizması.

6.2 `if` İfadesi — Temel

Mosh:

> *"ifadeler programlarımıza gerçekten yardımcı olabiliriz. Yapmak kararlar."* — Mosh (Türkçe dublaj), 1:40:18

6.2.1 Günlük Hayattan Örnek

```
"if hungry: eat breakfast"
"if cloudy: bring umbrella ELSE: bring sunglasses"
"if meat: order steak elif pasta: order spaghetti ELSE: order salad"
```

Programcı yapısı tıpatıp aynı:

```
if ac:
    kahvalti_yap()

if bulutlu:
    semsiye_getir()
else:
    gunes_gozlugu_getir()

if etyemek:
```

```
    biftek_siparis()
elif makarna:
    spagetti_siparis()
else:
    salata_siparis()
```

6.2.2 Sentaks

> *"Eğer bir if ifadesi kullanmak istersem tek yapmam gereken yazmak* ***if****. ve sonra Sadece yazmam gerek bir durum."* — Mosh (Türkçe dublaj), 1:44:22

```
if kosul:
    # girintili kod (kosul True ise calisir)
    print("kosul dogru!")
```

Dört parça:

- **`if`** — anahtar kelime.
- **koşul** — `True` veya `False` üreten bir ifade.
- **`:`** — gövde başlangıcı.
- **girintili kod** — koşul `True` olduğunda çalışan blok.

6.2.3 İlk Örnek

```
is_male = True

if is_male:
    print("Sen bir erkeksin.")

print("Bu satir her zaman calisir (girinti yok).")
```

`is_male` `False` yapılsa, `if` içindeki kod **atlanır**; gerisi normal çalışır.

6.2.4 Sanity Check Pattern

> Builder Notu — Üretim ML Kodu
>
> ML'in en yaygın if pattern'i: **sanity check + early termination**.

```python
import torch

def train_step(model, batch):
    loss = model(batch).loss

    if torch.isnan(loss):
        raise RuntimeError("Loss is NaN - training diverged!")

    if loss.item() > 1e6:
        print("Warning: loss is very large, check learning rate")

    return loss
```

Üç `if` — üç farklı sanity check. Üretim ML kodu bu pattern ile dolu. Mosh'un "if condition: act" temel yapısı, savunma mekanizması.

6.3 `else` ve `elif` — Çoklu Dallar

6.3.1 `else`

```python
is_male = True

if is_male:
    print("Sen bir erkeksin.")
else:
    print("Sen bir erkek degilsin.")
```

Mutlaka biri çalışır.

6.3.2 `elif`

```python
restoran_yemek = "makarna"

if restoran_yemek == "biftek":
    print("Biftek siparis edildi.")
elif restoran_yemek == "makarna":
    print("Spagetti ve kofte siparis edildi.")
elif restoran_yemek == "balik":
    print("Levrek siparis edildi.")
else:
    print("Salata siparis edildi.")
```

Akış:

1. `if` kontrol → False → atla.
2. İlk `elif` → True → **bu bloğu çalıştır + tüm zinciri bitir**.
3. Diğerleri çalışmaz.

Tek bir blok çalışır — koşullar `True` olsa bile sadece **ilki**.

6.3.3 Pratik — Not Sistemi

```
puan = 87

if puan >= 90:
    harf = "A"
elif puan >= 80:
    harf = "B"
elif puan >= 70:
    harf = "C"
elif puan >= 60:
    harf = "D"
else:
    harf = "F"
```

İlk `elif`'in koşulu yazılırken `87 < 90` zaten **biliniyor** (önceki `if` False olduğu için buradayız). Bu yüzden sadece alt sınırı yazmak yeterli.

6.3.4 Multi-Branch ML'de

Builder Notu — Confidence Dispatch

```
import torch

logits = model(x)
probs = torch.softmax(logits, dim=-1)
max_prob, pred_class = probs.max(dim=-1)

# Sklaer Python (yavaş):
for i in range(batch_size):
    if max_prob[i] > 0.9:
        action = "accept"
    elif max_prob[i] > 0.5:
        action = "review"
    elif max_prob[i] > 0.3:
        action = "human_review"
    else:
        action = "reject"

# Vektörlü PyTorch (hızlı):
actions = torch.zeros_like(max_prob, dtype=torch.long)
actions = torch.where(max_prob > 0.9, 0, actions)
actions = torch.where((max_prob > 0.5) & (max_prob <= 0.9), 1, actions)
actions = torch.where((max_prob > 0.3) & (max_prob <= 0.5), 2, actions)
actions = torch.where(max_prob <= 0.3, 3, actions)
```

`torch.where` Mosh'un `if/elif/else` zincirinin **tensor-element-wise** versiyonu. Tek hamlede tüm batch için karar verir.

6.4 Karşılaştırma Operatörleri

> *"ifadelerin kullanıp kullanılmadığını başka bir yolla inceleyeceğiz. Sadece boolean değerleri kullanmak yerine, aslında karşılaştırabiliriz."* — Mosh (Türkçe dublaj), 1:54:30

6.4.1 Altı Operatör

```
a = 5
b = 10

a == b   # False  (esit mi?)
a != b   # True   (esit DEGIL mi?)
a < b    # True   (kucuk mu?)
a > b    # False  (buyuk mu?)
```

```
a <= b   # True
a >= b   # False
```

Her karşılaştırma **boolean** döner.

6.4.2 KRİTİK Tuzak — = vs ==

```
a = 5        # ATAMA: a degiskenine 5 koy
a == 5       # KARSILASTIRMA: a 5'e esit mi? (True)

if a = 5:    # SyntaxError! atama if kosulu olamaz
    ...

if a == 5:   # OK
    ...
```

Python tasarımı bu hatayı **erken yakalar** (SyntaxError). C/Java'da `if (a = 5)` sessizce çalışır.

6.4.3 String Karşılaştırma

```
isim = "Deniz"

if isim == "Deniz":
    print("Selam Deniz!")

# Alfabetik:
"apple" < "banana"    # True   (a < b)
"apple" < "Apple"     # False  (A buyuk harf ASCII'de KUCUK)

# Case-insensitive:
if isim.lower() == "deniz":   # her ikisini küçük, sonra karsilastir
    print("Selam!")
```

6.4.4 Mosh'un Max Örneği

```
def max_num(num1, num2, num3):
    if num1 >= num2 and num1 >= num3:
        return num1
    elif num2 >= num1 and num2 >= num3:
        return num2
    else:
```

```
        return num3

print(max_num(3, 4, 5))      # 5
print(max_num(40, 4, 5))     # 40
```

6.4.5 Zincirleme — Python Özel

```
# C/Java tarzi:
if 0 < x and x < 10:
    print("0-10 arasinda")

# Python ozel - zincirleme:
if 0 < x < 10:
    print("0-10 arasinda")

# Matematik notasyonu:
if 18 <= yas < 65:
    print("Calisma yasinda")
```

C/Java/JS bu syntax'ı desteklemez.

6.4.6 Tensor Karşılaştırma

Builder Notu — Boolean Mask

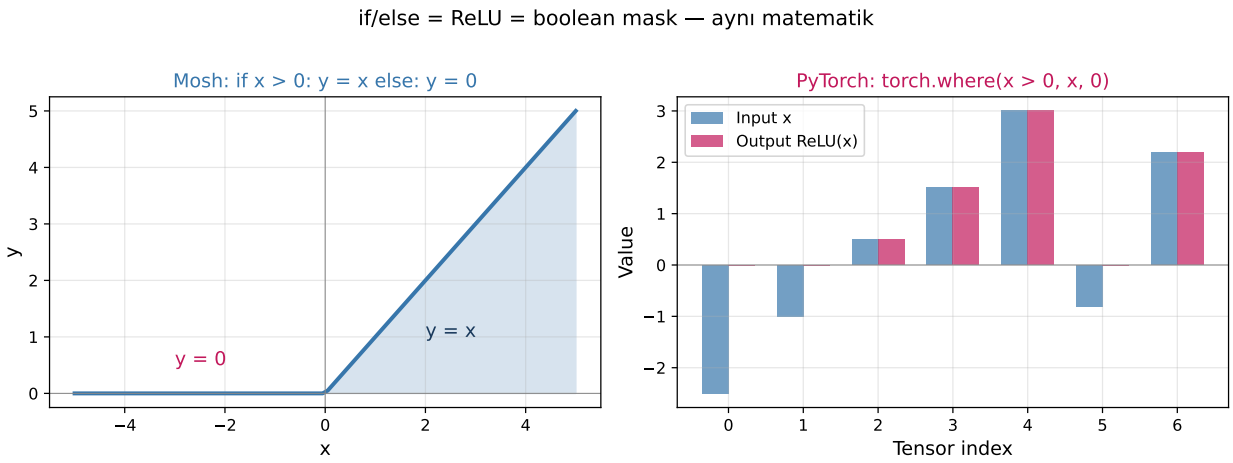
PyTorch'ta karşılaştırma operatörleri **element-wise**:

```
import torch

x = torch.tensor([1, 5, 3, 8, 2])
threshold = 4

mask = x > threshold
print(mask)             # tensor([False,  True, False,  True, False])
print(mask.dtype)       # torch.bool

# Mask ile filtreleme:
print(x[mask])          # tensor([5, 8])
print(mask.sum())       # tensor(2)  - kaç eleman threshold'u astı
```

Mosh'un `if x > 4:` skaler karşılaştırması, PyTorch'ta `x > 4` ile **tensor-of-booleans** üretir. Bu **mask** ML'in temel veri yapısı:

- **Attention mask** — hangi token'lara dikkat
- **Padding mask** — hangi pozisyonlar boş

- **Image segmentation** — hangi pixel obje
- **Loss masking** — hangi sample'lar hesaba katılsın



Şekil 6.2: Mosh'un if/else'i = ReLU = `torch.where(x>0, x, 0)` — aynı mantık

6.5 Mantık Operatörleri — `and`, `or`, `not`

6.5.1 `and` — İkisi de Doğru

```
is_male = True
is_tall = True

if is_male and is_tall:
    print("Uzun boylu bir erkeksin.")
```

`True and True` = `True`; başka her şey = `False`.

6.5.2 `or` — En Az Biri Doğru

```
if is_male or is_tall:
    print("Erkek veya uzun (veya ikisi).")
```

`False or False` = `False`; başka her şey = `True`.

6.5.3 `not` — Tersine Çevir

```
if not is_male:
    print("Erkek degilsin.")
```

6.5.4 Birleştirme

```
is_male = True
is_tall = False

if is_male and is_tall:
    print("Uzun boylu erkek.")
elif is_male and not is_tall:
    print("Kisa boylu erkek.")
elif not is_male and is_tall:
    print("Uzun boylu (erkek degil).")
else:
    print("Ne erkek ne de uzun.")
```

6.5.5 Operatör Önceliği

```
# not > and > or
True or False and False    # = True or (False and False) = True
not True and False         # = (not True) and False = False
```

Şüphedeysen **parantez koy**.

6.5.6 Conditional Expression (Ternary)

```
yas = 25

# Cok satirli:
if yas >= 18:
    durum = "yetiskin"
else:
    durum = "cocuk"

# Tek satirli:
durum = "yetiskin" if yas >= 18 else "cocuk"
```

ML kodda yaygın:

```
# Learning rate schedule:
lr = 1e-3 if epoch < 10 else 1e-4

# Device:
device = "cuda" if torch.cuda.is_available() else "cpu"
```

6.5.7 Tensor'larda &, |, ~

⚠ Builder Notu — Tensor Bitwise

PyTorch'ta `and`, `or`, `not` **tensor-geneninde** bitwise olarak:

```
import torch

mask1 = torch.tensor([True, False, True, True])
mask2 = torch.tensor([True, True, False, True])

mask1 & mask2    # tensor([ True, False, False,  True])  AND
mask1 | mask2    # tensor([ True,  True,  True,  True])  OR
~mask1           # tensor([False,  True, False, False])  NOT
```

Dikkat: `and`/`or`/`not` Python'un boolean operatörleri (skaler); `&`/`|`/`~` bitwise (tensor). Karıştırma:

```
# YANLIS:
if mask1 and mask2:    # RuntimeError - hangi True/False?
    ...

# DOGRU:
combined = mask1 & mask2     # element-wise

# Skaler kontrol:
if mask1.all():   # tum elementler True mi?
    ...
if mask1.any():   # en az bir True mi?
    ...
```

ML pattern — attention masking:

```
padding_mask = (tokens == PAD_TOKEN)
causal_mask = torch.tril(torch.ones(seq_len, seq_len)).bool()

combined_mask = padding_mask | ~causal_mask

attention_scores.masked_fill_(combined_mask, float('-inf'))
```

Bu transformer attention'ın çekirdek mekanizması. **GPT/BERT/Claude'un içinde** tam olarak böyle kullanılır.

6.6 Truthy / Falsy

Python her objeyi bir boolean değere indirgeyebilir.

Falsy (False gibi):

- `False`, `None`, `0`, `0.0`, `""` (boş string)
- `[]`, `()`, `{}`, `set()` (boş koleksiyonlar)

Truthy (her şey):

- `True`, sıfırdan farklı her sayı, boş olmayan her şey

6.6.1 Yaygın Pattern'lar

```
liste = []
if liste:
    print("Dolu")
else:
    print("Bos")           # bu çalışır

# Sayi sifir mi:
sayi = 0
if not sayi:
    print("Sifir veya hicbir sey")

# None kontrol (genelde `is None` daha iyi):
result = None
if result is None:
    print("Sonuc yok")
```

6.6.2 `is None` vs `== None`

```
# IYI:
if x is None:
    ...

# COGUNLUKLA OK ama özel durumlarda buggy:
if x == None:
    ...
```

`is` referans kıyası (aynı obje mi?); `==` değer kıyası. `None` singleton — sadece bir tane var.

6.6.3 ML'de Truthy/Falsy

```
if dataloader:                    # data var mi?
    for batch in dataloader:
        ...

if model is not None:
    model.eval()
```

```
if all_losses:                    # liste bos degilse
    avg = sum(all_losses) / len(all_losses)
else:
    avg = 0
```

`if liste:` empty kontrolü Python'un en sevilen deyimlerinden. C/Java'da `if (list != null && list.size() > 0)`; Python: `if liste:`.

6.7 Proje — Better Calculator

Ders 3'te basit calculator (toplama). Şimdi dört işlem:

```
num1 = float(input("Birinci sayi: "))
op = input("Islem (+, -, *, /): ")
num2 = float(input("Ikinci sayi: "))

if op == "+":
    print(num1 + num2)
elif op == "-":
    print(num1 - num2)
elif op == "*":
    print(num1 * num2)
elif op == "/":
    print(num1 / num2)
else:
    print("Bilinmeyen islem!")
```

Çalıştır:

```
Birinci sayi: 7
Islem (+, -, *, /): *
Ikinci sayi: 3
21.0
```

6.7.1 İyileştirme — Sıfıra Bölme

```
elif op == "/":
    if num2 == 0:
        print("Hata: sifira bolunemez")
    else:
        print(num1 / num2)
```

6.7.2 ML Pattern — Model Seçimi

Builder Notu — Factory Pattern

```
model_type = input("Model tipi (cnn/rnn/transformer): ")

if model_type == "cnn":
    model = build_cnn()
elif model_type == "rnn":
    model = build_rnn()
elif model_type == "transformer":
    model = build_transformer()
else:
    raise ValueError(f"Bilinmeyen: {model_type}")

model.train()
```

Mosh'un calculator'ı, ML'deki **factory pattern** ve **configuration-driven model building**'in mikro versiyonu. Hugging Face `AutoModel.from_pretrained("bert-base-uncased")` — içeride hangi mimari? `bert` ise BERT, `gpt` ise GPT — if/elif zinciri.

6.8 Bu Dersin Özeti

1. **`if kosul:`** — koşul True ise blok çalışır.
2. **`else`** — alternatif.
3. **`elif`** — çoklu dal; ilk True kazanır.
4. **`==`, `!=`, `<`, `>`, `<=`, `>=`** — boolean döner.
5. **`=` vs `==`** — atama vs karşılaştırma; karıştırma SyntaxError.
6. **Zincirleme `0 < x < 10`** — Python özel.
7. **`and`, `or`, `not`** — Python skaler.
8. **`&`, `|`, `~`** — tensor için bitwise.
9. **Truthy/Falsy** — `if liste:` empty check.
10. **`x if cond else y`** — tek satırlık conditional.

Tek Bir Cümle

`if/elif/else` Python'a karar verme yetisi kazandırır — Mosh'un "if hungry: eat else: skip" mantığı, PyTorch'taki `torch.where(x > 0, x, 0)` (ReLU!) ve transformer'lardaki attention masking'in (`padding_mask | ~causal_mask`) çekirdek atasıdır; karşılaştırma operatörleri tensor için **boolean mask** üretir — ML'in temel veri yapısı; ve `if liste:` boş kontrolü Python'un okunabilirlik felsefesinin en güzel örneklerinden biridir.

6.9 Egzersizler

Egzersiz 1. Yaş kategorileri:

- 0-2: “bebek”, 3-12: “cocuk”, 13-17: “genc”, 18-64: “yetiskin”, 65+: “yasli”.

```
yas = int(input("Yas: "))

if ???:
    print("bebek")
elif ???:
    ...
```

Sıra önemli — katıdan gevşeğe.

Egzersiz 2. Şifre güçlülük kontrolü (`and`):

```
sifre = input("Sifre: ")

uzun = len(sifre) >= 8
buyuk = any(c.isupper() for c in sifre)
rakam = any(c.isdigit() for c in sifre)

if uzun and buyuk and rakam:
    print("Sifre guclu!")
else:
    print("Eksik:")
    if not uzun: print("- En az 8 karakter")
    if not buyuk: print("- Buyuk harf")
    if not rakam: print("- Rakam")
```

Egzersiz 3. Calculator’a `%` ve `**` ekle + sıfıra bölme kontrolü:

```
num1 = float(input("Birinci sayi: "))
op = input("Islem (+, -, *, /, %, **): ")
num2 = float(input("Ikinci sayi: "))

if op == "+":
    sonuc = num1 + num2
elif ???:
    ...
```

Egzersiz 4. Truthy/Falsy tahmin et:

```
test = [0, 0.0, "", " ", [], [0], None, "False"]

for x in test:
    if x:
        print(f"{x!r} → TRUTHY")
    else:
        print(f"{x!r} → FALSY")
```

`"False"` (string) neden truthy? `[0]` neden truthy?

Egzersiz 5. (Builder eksen — tensor mask) `torch.where` mantığını Python listesi ile simüle et:

```
skorlar = [0.95, 0.42, 0.78, 0.31, 0.88, 0.55, 0.12, 0.99, 0.45, 0.67]

def aksiyon_belirle(skor):
    if ???:           # > 0.9 → "accept"
        return "accept"
    elif ???:         # > 0.5 → "review"
        return "review"
    else:
        return "reject"

aksiyonlar = [aksiyon_belirle(s) for s in skorlar]

for s, a in zip(skorlar, aksiyonlar):
    print(f"Skor {s:.2f} → {a}")

print(f"\nAccept: {aksiyonlar.count('accept')}")
print(f"Review: {aksiyonlar.count('review')}")
print(f"Reject: {aksiyonlar.count('reject')}")
```

Bu egzersiz, ML'de tensor üzerinde **vektörlü** olarak yapılan kararın Python liste üzerinde **döngülü** versiyonu.

6.10 Sonraki Ders İçin Hazırlık

Ders 7: Sözlükler ve Setler

Mosh'un Ch 19 (Dictionaries) + Set kavramı (Mosh göstermez ama Python ekosistemi şart). Liste sıralı koleksiyondu; dict **anahtar-değer eşlemesi**, set **unique elemanlar**.

- **Mosh'un Ch 19'unu izle** (2:07:20 - 2:14:16, ~7 dk).
- **Şu cümleyi içselleştir:** "Dict = anahtarla bulma; Set = unique elemanlar."

Bu dersten tek bir şey alıp gideceksen

`if/elif/else` Python'a karar verme yetisi kazandırır — Mosh'un "if hungry: eat else: skip" mantığı, modern ML'in en derin yapı taşıdır: PyTorch'un `torch.where(x > 0, x, 0)` (ReLU!), transformer attention masking (`padding_mask | ~causal_mask`), tüm sanity check'ler (`if loss.isnan(): raise`), tüm conditional schedule'lar (`lr = 1e-3 if epoch < 10 else 1e-4`) — hepsi bu temel mantığın doğal uzantısı.

7 Sözlükler ve Setler

`{k: v}` ve `{a, b}` — `state_dict`, vocab, data leakage check

Bölüm bilgisi

- **Mosh'un videosu:** Chapter 19 (Dictionaries) (7 dk) + **Mosh'un göstermediği:** Set
- **Bölüm aralığı:** 2:07:20 — 2:14:16 (+ Set bu derste eklendi)
- **Kaynaklar:** Python docs — dict · Python docs — set
- **Okuma süresi:** 35 dk

7.1 Bu Derste Ne Var?

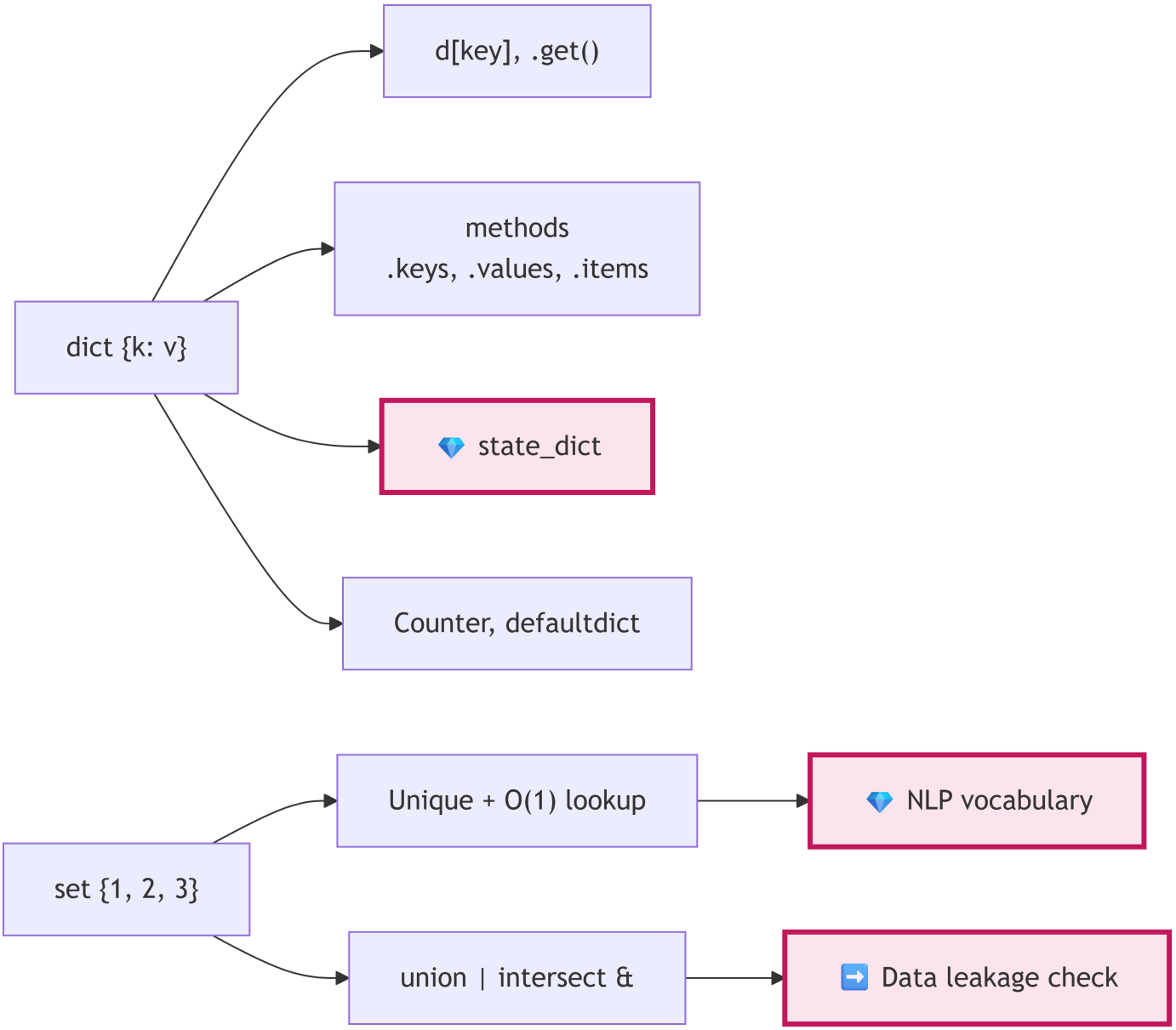
Python'un dört temel koleksiyon yapısının ikincisi ve üçüncüsünü öğreneceksin: **dictionary (dict)** ve **set**. Liste sıralı koleksiyondu (Ders 4); dict **anahtar-değer eşlemesi**, set ise **unique elemanların sırasız koleksiyonu**.

Dersin beş parçası:

1. **Dictionary (sözlük).** — `{"anahtar": "değer"}`. Mosh'un ay-isim dönüştürücüsü.
2. **Dict erişim ve metodlar.** — `dict[k]`, `.get()`, `.keys()`, `.values()`, `.items()`.
3. **Set (küme).** — `{1, 2, 3}`. **Unique** elemanlar.
4. **Set operasyonları.** — `|`, `&`, `-`, `^`.
5. **Dört koleksiyon karşılaştırması.** — list/tuple/dict/set.

Builder Notu — Bu Dersin ML Köprüleri

- **Dict = JSON config / hyperparameter config.** ML eğitim kodlarının %90'ında `config = {"lr": 1e-3, "batch_size": 64, "epochs": 100}`. JSON dosyaları zaten dict olarak yüklenir (`json.load(...)`).
- **Dict = PyTorch `model.state_dict()`.** Modelin tüm ağırlıkları ad-tensor map'i. Model kaydetme/yükleme bu dict üzerinden işler.
- **Dict = HuggingFace tokenizer vocab.** `tokenizer.vocab` her tokeni id'sine map'ler: `{"hello": 7592, "world": 2088, ...}`. LLM'lerin temeli bu dict'tir.
- **Set = unique label collection.** ML'de "kaç tane benzersiz sınıf var?" → `len(set(labels))`. Vocabulary oluşturma.
- **Set = duplicate filtering.** `list(set(items))` yinelenenleri ayıkla.
- **Set operations = data filtering venn.** Train set vs val set vs test set kelime kesişimi,



Şekil 7.1: Ders 7'nin akış haritası — dict ve set'in ML evreni

örütme kontrolü.
- `collections.Counter, defaultdict.` Standart `dict`'in güçlü kardeşleri. `Counter(labels)` ML class imbalance kontrolü için günlük pratik.

7.2 Dictionary — Anahtar-Değer Eşlemesi

Mosh:

> *"bir sözlük python'da bilgi depolamamıza izin veren özel bir yapıdır. ne denir anahtar değer çiftleri."* — Mosh (Türkçe dublaj), 2:07:31

7.2.1 Günlük Hayat Analojisi

Mosh kelimenin kendisinden faydalanıyor: "sözlük" gibi.

> *"kelime benzersiz bir şekilde tanımlayana benzer Sözlüğün içinde, ve sonra değer gerçek tanım olacaktır."* — Mosh (Türkçe dublaj), 2:08:02

- **Kelime** → anahtar (key).
- **Tanım** → değer (value).

7.2.2 Motivasyon — Ay Kısaltma Dönüştürücüsü

List ile (kötü):

```
kisaltmalar = ["Jan", "Feb", "Mar", ...]
tam_isimler = ["January", "February", "March", ...]

idx = kisaltmalar.index("Mar")
print(tam_isimler[idx])
```

İki ayrı liste, indekslerin denkleşmesini elle sağlaman gerek. Kırılgan.

Dict ile (iyi):

```
ay_donusumleri = {
    "Jan": "January",
    "Feb": "February",
    "Mar": "March",
}

print(ay_donusumleri["Mar"])    # March
```

7.2.3 Sentaks

> *"açık ve kapalı kıvrımlı parantez"* — Mosh (Türkçe dublaj), 2:09:08

```
sozluk_adi = {
    anahtar1: deger1,
    anahtar2: deger2,
    anahtar3: deger3,
}
```

Küme parantezi `{ }` ile. Her giriş `anahtar: değer`. Aralarında virgül.

7.2.4 Anahtar Kuralları

> *"anahtarların hepsinin benzersiz olması gerekiyor."* — Mosh (Türkçe dublaj), 2:10:15

Üç kural:

1. **Unique** — aynı anahtar iki kere yazılamaz.
2. **Immutable tip** — anahtar `str`, `int`, `float`, `tuple`, `bool`, `frozenset` olabilir. List/dict/set **olamaz**.
3. **Hashable** — yukarıdaki kısıtlamanın teknik adı.

```
# OK:
d = {"key": 1}                  # str
d = {42: "answer"}              # int
d = {(3, 5): "point"}           # tuple (Ders 4'te bu nedenle tuple gerekiyordu!)

# YANLIS:
d = {[1, 2]: "list"}            # TypeError: unhashable
```

7.2.5 Dict ML'in DNA'sı

> **Builder Notu — 10 ML Pattern**
>
> ML kodunda dict 10+ farklı yerde:

```python
# 1. Hyperparameter config:
config = {"lr": 1e-3, "batch_size": 64, "device": "cuda"}

# 2. JSON yükleme:
import json
with open("config.json") as f:
    cfg = json.load(f)              # cfg bir dict!

# 3. Model state_dict:
import torch
state = model.state_dict()           # OrderedDict[str, Tensor]
torch.save(state, "model.pt")

# 4. Tokenizer vocab:
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
vocab = tokenizer.get_vocab()        # dict[str, int]
print(vocab["hello"])                # 7592

# 5. Dataset sample:
sample = {"image": tensor, "label": 7, "path": "img.jpg"}

# 6. Metric tracking:
metrics = {"loss": [], "acc": [], "f1": []}

# 7. Model outputs (HuggingFace):
outputs = model(input_ids)
print(outputs.logits)                # dict-like cikti
```

Mosh'un "ay-isim dönüştürücüsü" mikro örneği, yukarıdaki tüm pattern'ların çekirdeği. **Tek bir kavram, on farklı görünüm.**

7.3 Dict Erişim — [] ve .get()

7.3.1 Bracket Access

```python
ay_donusumleri = {
    "Jan": "January",
    "Feb": "February",
    "Nov": "November",
}

ay_donusumleri["Nov"]    # 'November'
ay_donusumleri["Luc"]    # KeyError: 'Luc'
```

7.3.2 `.get()` — Güvenli Erişim

> *"get işlevini kullanmanın nesi harika Aslında kullanmak istediğim varsayılan bir değer belirtebilirim. eğer bu anahtarlar bulamazsa"* — Mosh (Türkçe dublaj), 2:12:22

```
ay_donusumleri.get("Nov")                # 'November'
ay_donusumleri.get("Luc")                # None (hata yok!)
ay_donusumleri.get("Luc", "Bilinmeyen")  # 'Bilinmeyen'
```

Pratik:

```
# Sert (anahtar olmalı):
lr = config["learning_rate"]

# Yumusak (default ile):
lr = config.get("learning_rate", 1e-3)
```

ML hyperparameter okuma çoğu zaman `.get(...)` ile default'lu.

7.3.3 Ekle / Değiştir / Sil

```
kullanici = {"ad": "Deniz", "yas": 35}

kullanici["sehir"] = "Istanbul"      # ekle
kullanici["yas"] = 36                # değiştir
del kullanici["sehir"]               # sil
yas = kullanici.pop("yas")           # sil ve değeri al
```

7.3.4 `in` Üyelik

```
"Jan" in ay_donusumleri    # True
"XYZ" in ay_donusumleri    # False
```

`in` dict için **anahtarı** kontrol eder.

7.4 Dict Metodları

```
ay = {"Jan": "January", "Feb": "February", "Mar": "March"}

# Anahtarlar / değerler / çiftler:
ay.keys()       # dict_keys(['Jan', 'Feb', 'Mar'])
ay.values()     # dict_values(['January', 'February', 'March'])
ay.items()      # dict_items([('Jan', 'January'), ...])

# Iteration:
for anahtar, deger in ay.items():
    print(f"{anahtar} -> {deger}")

# Birleştir:
a = {"x": 1, "y": 2}
b = {"y": 99, "z": 3}
a.update(b)     # a artik {"x": 1, "y": 99, "z": 3}

# Modern (Python 3.9+):
c = a | b       # yeni dict
```

7.4.1 Dict Comprehension

```
kareler = {x: x**2 for x in range(1, 6)}
# {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

kullanici = {"ad": "Deniz", "yas": 35}
buyuk_dict = {k.upper(): str(v).upper() for k, v in kullanici.items()}
```

7.5 Counter ve defaultdict — Dict'in Güçlü Kardeşleri

7.5.1 Counter — Sayma

```
from collections import Counter

labels = ["kedi", "kopek", "kedi", "kus", "kedi", "kopek"]
counter = Counter(labels)
# Counter({'kedi': 3, 'kopek': 2, 'kus': 1})

counter.most_common(2)
# [('kedi', 3), ('kopek', 2)]
```

ML class imbalance kontrolü:

```
from collections import Counter

class_counts = Counter(dataset_labels)

for cls, count in class_counts.most_common():
    pct = count / len(dataset_labels) * 100
    print(f"  {cls}: {count} ({pct:.1f}%)")
```

7.5.2 `defaultdict` — Otomatik Initialize

```
from collections import defaultdict

# Normal dict (uzun):
gruplar = {}
for n in [1, 2, 3, 4, 5]:
    grup = "cift" if n % 2 == 0 else "tek"
    if grup not in gruplar:
        gruplar[grup] = []
    gruplar[grup].append(n)

# defaultdict (kısa):
gruplar = defaultdict(list)
for n in [1, 2, 3, 4, 5]:
    grup = "cift" if n % 2 == 0 else "tek"
    gruplar[grup].append(n)
```

7.6 Set — Unique Elemanlar

Mosh göstermez ama Python ekosisteminin temel parçası.

7.6.1 Set Nedir?

Set: sırasız + unique elemanlar.

```
sayilar = {1, 2, 3, 4, 5}
print(type(sayilar))  # <class 'set'>

karisik = {1, 2, 2, 3, 3, 3, 4}
print(karisik)        # {1, 2, 3, 4}  - yinelenenler kayboldu
```

Liste ile farklar:

- **Sıra yok** — `set[0]` çalışmaz.
- **Unique** — yinelenenler otomatik filtrelenir.

7.6.2 Sentaks Tuzağı

```
# Boş set:
bos = set()        # bos set
yine_bos = {}      # boş DICT! (set değil)

# Dolu:
sayilar = {1, 2, 3}

# Liste'den (deduplication):
from_list = set([1, 2, 2, 3, 3, 4])   # {1, 2, 3, 4}
```

`{}` boş **dict**, `set()` boş **set** — Python tasarım kararı.

7.6.3 Set Metodları

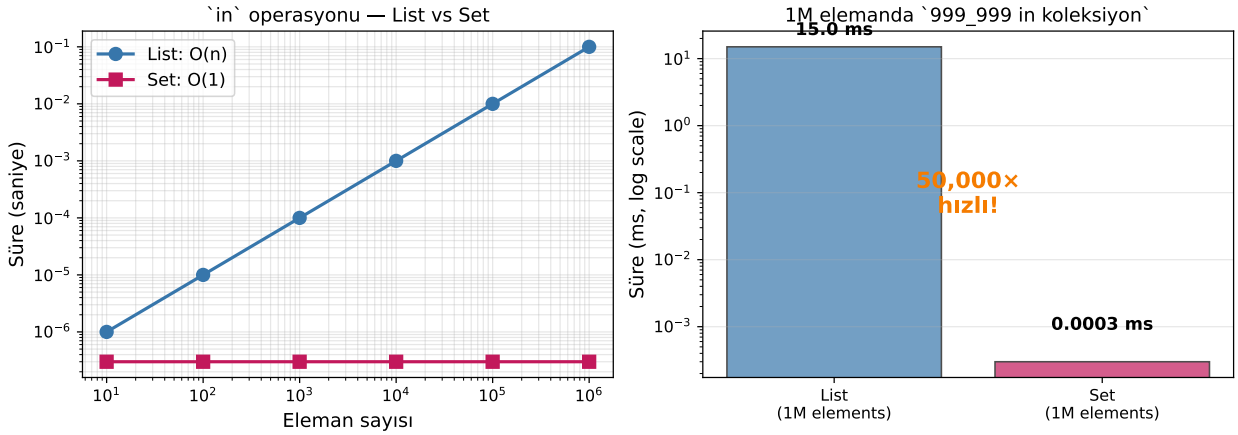
```
sayilar = {1, 2, 3}

sayilar.add(4)         # {1, 2, 3, 4}
sayilar.add(2)         # zaten var, hicbir sey olmaz
sayilar.remove(2)      # {1, 3, 4}
sayilar.discard(100)   # OK (remove → KeyError)
n = sayilar.pop()      # rastgele eleman çıkar

len({1, 2, 3, 4, 5})   # 5
3 in {1, 2, 3}         # True
```

7.6.4 Set'in Süper Gücü — `in` O(1)

```
import time

n = 1_000_000
liste = list(range(n))
kume = set(liste)

# Liste'de arama: O(n)
start = time.time()
print(n - 1 in liste)
print(f"List: {time.time() - start:.4f}s")
# 0.0150s

# Set'te arama: O(1)
start = time.time()
```

```
print(n - 1 in kume)
print(f"Set: {time.time() - start:.7f}s")
# 0.0000003s
```

50,000 kat hızlı. Set hash table kullanır.



Şekil 7.2: Set vs List performans — `in` operasyonu için 50K kat hız farkı

7.7 Set Operasyonları — Küme Matematiği

```
a = {1, 2, 3, 4, 5}
b = {4, 5, 6, 7, 8}

a | b    # {1, 2, 3, 4, 5, 6, 7, 8}   birleşim
a & b    # {4, 5}                       kesişim
a - b    # {1, 2, 3}                    fark
a ^ b    # {1, 2, 3, 6, 7, 8}           simetrik fark

# Alt-küme:
{1, 2} <= {1, 2, 3}    # True
{1, 2} <  {1, 2}       # False (esit, oz degil)
```

7.7.1 Vocabulary Oluşturma (NLP)

Builder Notu — NLP Vocab

```
documents = [
    "the cat sat on the mat",
    "the dog ate the bone",
    "a cat and a dog",
]

vocab = set()
for doc in documents:
    vocab.update(doc.split())

print(f"Vocab size: {len(vocab)}")    # 10
```

7.7.2 Train/Val/Test Data Leakage Check

Builder Notu — Data Leakage

Üretim ML'inde kritik kontrol:

```
# Test ID'leri train'de var mı? (data leakage)
train_ids = set(train_df["id"])
test_ids = set(test_df["id"])

leakage = train_ids & test_ids
if leakage:
    print(f"WARNING: {len(leakage)} ID train+test'te")

# Val'da train'de görünmemiş kelimeler:
train_vocab = set(...)
val_vocab = set(...)
unseen_val = val_vocab - train_vocab
```

Milyon kayıt için saniyede biter.

7.8 Dört Koleksiyon — Karşılaştırma

Özellik	list `[1, 2]`	tuple `(1, 2)`	dict `{a: 1}`	set `{1, 2}`
Sıralı	Evet	Evet	Evet (3.7+)	Hayır
Değişebilir	Evet	**Hayır**	Evet	Evet
Yinelenen OK	Evet	Evet	Anahtar değil	**Hayır**
Erişim	`l[i]`	`t[i]`	`d[key]`	`in`

Özellik	list `[1, 2]`	tuple `(1, 2)`	dict `{a: 1}`	set `{1, 2}`
Hashable mi?	Hayır	**Evet**	Hayır	Hayır (frozenset evet)
Sentaks	`[a, b]`	`(a, b)`	`{a: 1}`	`{a, b}`

7.8.1 Karar Akışı

```
Sıralı mı olmalı?
   HAYIR → Set (unique + üyelik kontrolü hızlı)
   EVET:
       Anahtar-değer mi?
           EVET → Dict
           HAYIR:
               Değişecek mi?
                   EVET → List
                   HAYIR → Tuple
```

7.8.2 Pratik

```python
# List:
losses = [0.5, 0.4, 0.3]            # epoch sirasi onemli
friends = ["Kevin", "Karen"]         # eklenebilir

# Tuple:
input_shape = (3, 224, 224)          # tensor boyutu
rgb = (255, 0, 0)                    # sabit

# Dict:
config = {"lr": 1e-3, "batch_size": 64}

# Set:
seen_ids = {1, 5, 23, 7}             # bir id gorduk mu?
```

7.9 Bu Dersin Özeti

1. **Dict (`{}`):** anahtar-değer eşlemesi. Anahtarlar **unique** + **hashable**.
2. **Erişim:** `d[k]` sert, `.get(k, default)` yumuşak.
3. **Iteration:** `.keys()`, `.values()`, `.items()`.
4. **`in`** dict için anahtarı kontrol.
5. **Set (`{...}` / `set()`):** unique + sırasız + hızlı `in`.
6. **Set ops:** `|`, `&`, `-`, `^`.
7. **`{}` boş dict, `set()` boş set.**
8. **`Counter`, `defaultdict`** — `collections` güçlü kardeşler.

9. **ML eki:** dict her yerde (config/state_dict/vocab), set vocab+dedup+filter için kritik.

> **!** Tek Bir Cümle
>
> Dict (`{anahtar: değer}`) anahtarla hızlı arama yapar — Mosh'un ay-isim örneği aslında PyTorch'taki `model.state_dict()`, HuggingFace tokenizer vocab'ı, ve JSON config dosyalarının ata-formudur; set (`{1, 2, 3}`) unique elemanları sırasız tutar ve `in` kontrolünü O(1)'e indirir — NLP vocabulary, label dedup, ve train/val data leakage kontrolünün günlük aracı.

7.10 Egzersizler

Egzersiz 1. Öğrenci sicil dict — 5 öğrenci, notlar list:

```
sicil = {
    "Deniz": [85, 90, 78],
    ???
}

# (a) "Aylin" ekle, notları [95, 88, 92]
# (b) "Deniz"in notuna 80 ekle
# (c) "Mosh"un ortalaması
# (d) Tüm ortalamalar
```

Egzersiz 2. Kelime sayma — manuel + Counter:

```
cumle = "the quick brown fox jumps over the lazy dog the dog naps"

# Manuel:
kelime_sayilari = {}
for kelime in cumle.split():
    kelime_sayilari[kelime] = kelime_sayilari.get(kelime, 0) + 1

# Counter (tek satır):
from collections import Counter
sayilar = Counter(cumle.split())
```

Egzersiz 3. Liste kesişim/fark — set ile:

```
a = [1, 2, 3, 4, 5, 6]
b = [4, 5, 6, 7, 8, 9]

kesisim_set = set(a) & set(b)
fark_set = set(a) - set(b)

print(kesisim_set)   # {4, 5, 6}
print(fark_set)      # {1, 2, 3}
```

1 milyon eleman için hangisi pratik?

Egzersiz 4. ML config güvenli erişim:

```
config = {
    "model": "resnet18",
    "lr": 1e-3,
    "batch_size": 64,
}

# Zorunlu:
model_name = config["model"]
lr = config["lr"]

# Opsiyonel:
epochs = config.get(???)        # default 100
device = config.get(???)        # default "cpu"
weight_decay = config.get(???)  # default 0.0
```

Egzersiz 5. (Builder eksen — model state JSON)

```
import json

model_state = {
    "layer1.weight": [[0.1, 0.2], [0.3, 0.4]],
    "layer1.bias": [0.0, 0.0],
    "layer2.weight": [[0.5, 0.6], [0.7, 0.8]],
    "layer2.bias": [0.1, 0.1],
}

# Kaydet:
with open("model.json", "w") as f:
    json.dump(model_state, f)

# Yükle:
with open("model.json") as f:
    loaded = json.load(f)

# Anahtar-shape print:
for name, weights in loaded.items():
    if isinstance(weights[0], list):
        shape = (len(weights), len(weights[0]))
    else:
        shape = (len(weights),)
    print(f"  {name}: shape={shape}")
```

PyTorch `torch.save` / `torch.load` mekanizmasının **JSON versiyonu**.

7.11 Sonraki Ders İçin Hazırlık

Ders 8: Döngüler

Bu kursun **en kritik dersi**. Mosh dört chapter'da (While, Guessing Game, For, 2D Lists). `for batch in dataloader:` modern AI'nın ana motoru.

- **Mosh'un Ch 20-22 ve Ch 24'ünü izle** (2:14:16-2:52:44, ~38 dk).
- **Şu cümleyi içselleştir:** "Döngü = tekrar etme = bir grup veri üzerinde aynı işi yapma."

Bu dersten tek bir şey alıp gideceksen

Dict (`{anahtar: değer}`) anahtarla hızlı arama yapar — Mosh'un ay-isim örneği aslında PyTorch'taki `model.state_dict()`, HuggingFace tokenizer vocab'ı, ve JSON config dosyalarının ata-formudur; set (`{1, 2, 3}`) unique elemanları sırasız tutar ve `in` kontrolünü O(1)'e indirir — NLP vocabulary, label dedup, ve train/val data leakage kontrolünün günlük aracı; Python'un dört koleksiyon yapısı (list, tuple, dict, set) artık tam senin elinde, ve modern ML kodunun her satırı bu dördünden birinin üzerinde inşa edilmiş.

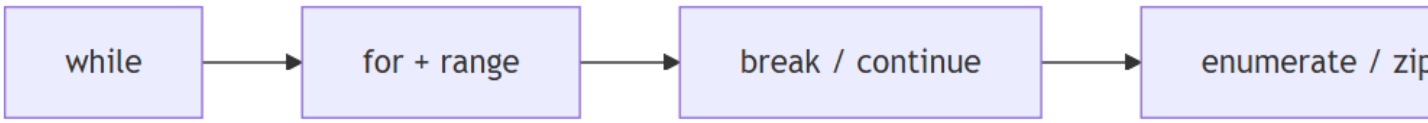
8 Döngüler

`for batch in dataloader` — modern AI'nın ana motoru

> **Bölüm bilgisi**
>
> - **Mosh'un videosu:** Chapters 20-22 + 24 (While → Guessing → For → 2D) (38 dk)
> - **Bölüm aralığı:** 2:14:16 — 2:52:44
> - **Kaynaklar:** Python docs — control flow · Python docs — range · Python docs — itertools
> - **Okuma süresi:** 60 dk

8.1 Bu Derste Ne Var?

Bu kursun **en kritik dersi**. Mosh dört chapter'da Python'un tekrar etme yetisini öğretiyor; biz bunu **modern ML kodunun ana motoru**na bağlıyoruz. `for batch in dataloader:` satırı olmasaydı, sinir ağları eğitilmez, LLM'ler üretilmez, modern AI olmazdı.



Şekil 8.1: Ders 8'in akış haritası — döngülerden ML eğitim motoruna

Dersin yedi parçası:

1. **`while` döngüsü.** — Koşul True iken tekrar.
2. **Guessing Game (proje).** — `while` + `if`.
3. **`for` döngüsü.** — Koleksiyon gezme.
4. **`range()`.** — Sayı dizisi.
5. **`break` ve `continue`.** — Akış kontrol.
6. **Nested loops + 2D listeler.** — Matris işleme.
7. **ML eğitim döngüsü.** — Mosh'un öğrettiklerinin AI'da uygulanması.

Builder Notu — Bu Dersin ML Köprüleri (EN GÜÇLÜ)

- **`for batch in dataloader` = ML eğitim döngüsünün ANA MOTORU.** Her sinir ağı eğitimi, her LLM fine-tune, her görüntü sınıflandırma — hepsi bu satırla çalışır. Mosh'un "döngüde her öğeye bir şey yap" sezgisi, milyarlarca parametre üstünde milyonlarca veri örneğini işleyen ML pipeline'ının çekirdek pattern'ı.
- **`for epoch in range(EPOCHS)` = dış eğitim döngüsü.** Bir epoch = veri setinin tamamen bir kez işlenmesi.
- **`while` = early stopping kontrolü.** `while patience_counter < max_patience: train_one_epoch()`.
- **`break` = early termination.** `if val_loss > best_loss: break`.
- **`continue` = bad batch skip.** `if torch.isnan(loss): continue`.
- **`enumerate` = batch index + batch.** `for i, batch in enumerate(loader):` logging için.
- **`zip` = paralel iteration.** `for img, label in zip(images, labels):`.
- **Nested for = batch × feature processing.** Python loop'ta yavaş, vektörlü yap'la **tek hamlede**.
- **List comprehension = vectorized construction.** `losses = [train(b) for b in batches]`.

8.2 Neden Döngü?

Şimdiye kadar yazdığımız kodun büyük eksiği: **tekrar yok**. Öğrenci sicil dict'imiz üç öğrenci tutarsa her birine elle erişmek zorunda kalıyoruz. 100 öğrencide kopya-yapıştır cehennem.

> *"süre döngü temelde python içinde bir döngüdür. ve yürüt Birden çok kez bir kod bloğu."*
> — Mosh (Türkçe dublaj), 2:14:16

Döngü üç güç sağlar:

1. **Tekrar** — aynı kodu N kez çalıştır.
2. **Gezme** — bir koleksiyondaki tüm öğelere ulaş.
3. **Birikim** — sonuçları toplama, sayma, filtreleme.

8.2.1 İki Tür

Tür	Ne zaman?
`while`	Koşula bağlı — bilinmeyen iterasyon sayısı
`for`	Bilinen koleksiyon — her elemanı sırayla

ML'de:

- `for batch in dataloader:` (her batch) → **`for`**
- `for epoch in range(100):` (100 epoch) → **`for`**
- `while not converged:` → **`while`**

- `while patience > 0:` → **`while`**

8.3 `while` Döngüsü

> *"buna döngü olarak bakın bekçi. ve temelde kod boyunca döngü devam edeceğiz while döngüsü içinde. bu durum doğru olduğu sürece."* — Mosh (Türkçe dublaj), 2:15:36

8.3.1 Sentaks

```
while kosul:
    # girintili kod (kosul True ise, sonra TEKRAR kontrol)
```

Kritik fark `if`'ten: `while` kodu **çalıştırıp koşulu tekrar kontrol eder**. `if` bir kere.

8.3.2 İlk Örnek

```
i = 1

while i <= 10:
    print(i)
    i += 1   # "i = i + 1" kisayolu

print("Dongu bitti")
```

Çıktı: `1, 2, 3, ..., 10, Dongu bitti`.

8.3.3 += Kısayollar

```
i = 5

i += 1    # i = i + 1   -> 6
i -= 2    # i = i - 2
i *= 3    # i = i * 3
i //= 2   # i = i // 2
i %= 2    # i = i % 2
```

8.3.4 Sonsuz Döngü Tuzağı

```
i = 1
while i <= 10:
    print(i)
    # i += 1 unutuldu! i hep 1, sonsuz dongu!
```

PyCharm'da `Ctrl+C` ile dur. **Bilinçli sonsuz döngü** (`while True:` + `break`) ML'de yaygın.

8.3.5 `while` = Early Stopping

Builder Notu — Patience Pattern

```
import torch

model = ...
optimizer = ...
best_val_loss = float('inf')
patience = 5
patience_counter = 0
epoch = 0
max_epochs = 1000

while patience_counter < patience and epoch < max_epochs:
    train_one_epoch(model, optimizer)
    val_loss = validate(model)

    if val_loss < best_val_loss:
        best_val_loss = val_loss
        patience_counter = 0   # iyilesti, sabir resetle
    else:
        patience_counter += 1  # iyilesmedi, sabir azalt

    epoch += 1

print(f"Egitim {epoch} epoch sonra bitti")
```

Mosh'un `i <= 10` sayı mantığı, ML'de "sabır kalması" koşuluna dönüşür.

8.4 Proje — Guessing Game

> *"bir süre döngü denilen bir şey [...] sürekli olarak kişiye sormak için sözcüğü tahmin etmek doğru tahmin edene kadar."* — Mosh (Türkçe dublaj), 2:21:47

8.4.1 Basit Sürüm

```
secret_word = "draft"
guess = ""

while guess != secret_word:
    guess = input("Tahmin et: ")

print("Kazandiniz!")
```

Problem: sonsuz tahmin yapabilir.

8.4.2 Üç Hak Sınırlı

```
secret_word = "draft"
guess_count = 0
guess_limit = 3

for attempt in range(guess_limit):
    guess = input(f"Tahmin et ({guess_limit - attempt} hak): ")
    if guess == secret_word:
        print("Kazandiniz!")
        break
else:
    # for-else: for normal bittiyse (break olmadan) calisir
    print("Kaybettiniz!")
```

`for-else` — Python'a özgü tuhaf ama güzel: `for` döngüsü `break` ile bitmediyse `else` çalışır.

8.4.3 Guessing Game = ML Hyperparameter Search

> Builder Notu — Optimization Loop
>
> Mosh'un üç-hak oyunu, ML hyperparameter search'ün mikro modeli:

```python
import random

best_loss = float('inf')
attempts = 0
max_attempts = 50

while attempts < max_attempts and best_loss > target_loss:
    config = {
        "lr": 10 ** random.uniform(-5, -1),
        "batch_size": random.choice([16, 32, 64, 128]),
        "hidden_dim": random.choice([64, 128, 256, 512]),
    }

    val_loss = train_and_evaluate(config)

    if val_loss < best_loss:
        best_loss = val_loss
        best_config = config

    attempts += 1
```

Mosh'un guessing game'i = AutoML/Optuna'nın **felsefi temeli**.

8.5 `for` Döngüsü

> *"bir for döngüsü python'da bize izin veren özel bir döngü türüdür farklı koleksiyonlar üzerinde dolaşmak"* — Mosh (Türkçe dublaj), 2:32:50

8.5.1 Sentaks

```python
for degisken in koleksiyon:
    # her oge icin tekrarlanir
    print(degisken)
```

8.5.2 String, List, Tuple, Set, Dict

```python
# String:
for letter in "Draft Academy":
    print(letter)

# List:
```

```
friends = ["Kevin", "Karen", "Jim"]
for friend in friends:
    print(f"Merhaba {friend}!")

# Tuple:
for c in (3, 5, 7):
    print(c)

# Dict:
kullanici = {"ad": "Deniz", "yas": 35, "sehir": "Istanbul"}
for key, value in kullanici.items():
    print(f"{key}: {value}")
```

`while` ile aynı işi yapmak için index yönetmek gerekir; `for` çok daha **Pythonic**.

8.5.3 Yararlı Helpers — `enumerate`, `zip`

```
# enumerate - index + eleman:
friends = ["Kevin", "Karen", "Jim"]
for i, friend in enumerate(friends):
    print(f"{i}: {friend}")
# 0: Kevin
# 1: Karen
# 2: Jim

# zip - paralel iteration:
isimler = ["Deniz", "Aylin", "Mosh"]
yaslar = [35, 28, 40]
for ad, yas in zip(isimler, yaslar):
    print(f"{ad} ({yas})")

# reversed / sorted:
for friend in reversed(friends):
    print(friend)

for friend in sorted(friends):
    print(friend)
```

8.6 `range()`

```
list(range(5))          # [0, 1, 2, 3, 4]
list(range(2, 8))       # [2, 3, 4, 5, 6, 7]
```

```
list(range(0, 20, 3))    # [0, 3, 6, 9, 12, 15, 18]
list(range(10, 0, -1))   # [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

`range(stop)` — `stop` **dahil değil** (yarı açık aralık).

8.6.1 `for` ile Kullanım

```
for i in range(5):
    print(i)

# 100 kez tekrar (sadece tekrar):
for _ in range(100):
    print("merhaba")    # 100 kez basilir
```

`_` — "değişkene ihtiyacım yok, sadece tekrar etmeliyim".

8.6.2 `range` vs `enumerate`

```
friends = ["Kevin", "Karen", "Jim"]

# C/Java tarzi (Python'da KOTU):
for i in range(len(friends)):
    print(f"{i}: {friends[i]}")

# Pythonic:
for i, friend in enumerate(friends):
    print(f"{i}: {friend}")
```

PEP 8 + linter'lar `enumerate` öneriri.

8.6.3 `range` ML'in Epoch Döngüsü

! Builder Notu — ML EĞİTİM OMURGASI

```
import torch
import torch.nn as nn

EPOCHS = 100
BATCH_SIZE = 64

model = ...
optimizer = ...

for epoch in range(EPOCHS):                    # outer for - epoch sayisi
    epoch_losses = []

    for batch_idx, batch in enumerate(train_loader):   # inner for + enumerate
        loss = compute_loss(batch)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        epoch_losses.append(loss.item())

        if batch_idx % 100 == 0:               # mod ile periyodik log
            print(f"Epoch {epoch}, Batch {batch_idx}: loss={loss.item():.4f}")

    avg_loss = sum(epoch_losses) / len(epoch_losses)
    print(f"Epoch {epoch} avg loss: {avg_loss:.4f}")
```

Mosh'un `for i in range(10):` mikro örneği ML'in `for epoch in range(EPOCHS):` makro hali. **Aynı yapı, daha büyük ölçek.**
Her ML eğitim sistemi bu iki nested for döngüsünden oluşur:

- **Outer:** epochs üzerinde (`range`)
- **Inner:** batch'ler üzerinde (`dataloader`)

8.7 break ve continue

```
# break - donguden tamamen cik:
for i in range(10):
    if i == 5:
        break
    print(i)
# 0, 1, 2, 3, 4

# continue - bu iterasyonu atla:
```

```python
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
# 1, 3, 5, 7, 9
```

8.7.1 ML Eşdeğerleri

```python
# break = early stopping:
for epoch in range(MAX_EPOCHS):
    if val_loss > best_loss * 1.1:
        print("Early stopping!")
        break

# continue = bad batch skip:
for batch in train_loader:
    loss = compute_loss(batch)
    if torch.isnan(loss):
        print("Bad batch, atlandi")
        continue
    loss.backward()
    optimizer.step()
```

8.8 Nested Loops + 2D Listeler

```python
matris = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
]

print(matris[0])       # [1, 2, 3]
print(matris[0][2])    # 3
print(matris[2][0])    # 7

# Nested for - her elemani yazdir:
for satir in matris:
    for eleman in satir:
        print(eleman, end=" ")
    print()
# 1 2 3
# 4 5 6
# 7 8 9
```

8.8.1 Çarpım Tablosu

```
for i in range(1, 6):
    for j in range(1, 6):
        print(f"{i*j:3d}", end=" ")
    print()
```

8.8.2 Nested = Çarpımsal Maliyet

```
# 100 x 100 = 10K iterasyon
# 1000 x 1000 = 1M iterasyon
# 1000 x 1000 x 1000 = 1B (tehlikeli!)
```

8.8.3 Vectorization Kararı

⚠ Builder Notu — Nested vs Tensor

Python nested loops kavramsal olarak doğru ama **performans için yetersiz**:

```python
import time
import numpy as np
import torch

A = [[i*j for j in range(1000)] for i in range(1000)]   # 1000x1000

# Python loops:
start = time.time()
result = [[x ** 2 for x in row] for row in A]
print(f"Python: {time.time() - start:.3f}s")
# ~0.5 sn

# NumPy:
A_np = np.array(A)
start = time.time()
result_np = A_np ** 2
print(f"NumPy: {time.time() - start:.6f}s")
# ~0.003 sn  - 100+ kat

# PyTorch GPU:
A_t = torch.tensor(A).cuda()
start = time.time()
result_t = A_t ** 2
torch.cuda.synchronize()
print(f"PyTorch GPU: {time.time() - start:.6f}s")
# 1000+ kat
```

ML perspektifi:

- `for batch in dataloader`: **OK** çünkü batch sayısı az (~1000).
- `for pixel in image`: **YASAK** çünkü pixel sayısı çok (224×224×3 150K). Tensor op ile yap.

8.9 List Comprehension

Mosh göstermez ama modern Python'un en sevdiği deyim.

```python
# Klasik for:
kareler = []
for i in range(10):
    kareler.append(i * i)

# List comprehension:
kareler = [i * i for i in range(10)]
```

```
# Filtreli:
cift_kareler = [i * i for i in range(10) if i % 2 == 0]

# Dict comprehension:
kareler_dict = {i: i*i for i in range(5)}

# Set comprehension:
benzersiz_kareler = {x*x for x in [-2, -1, 0, 1, 2]}
```

8.9.1 Comprehension ML'de

```
# Bir liste tensoru cevirme:
tensors = [torch.tensor(data) for data in batches]

# Filtering:
valid_losses = [loss for loss in all_losses if not torch.isnan(loss)]

# Dict olusturma:
param_count = {name: p.numel() for name, p in model.named_parameters()}
```

3 satırı tek satıra indirir. Okunabilir ve hızlı.

8.10 ML Eğitim Döngüsü — KURSUN ZIRVESI

Bu bölüm Ders 8'in **ve tüm kursun** zirvesi. Mosh'un öğrettikleri tek bir kod parçasında bir araya geliyor. **Bu kod, modern AI'nın kalbi.**

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader

# Ders 7 (dict + config):
config = {
    "lr": 1e-3,
    "batch_size": 64,
    "epochs": 100,
    "patience": 5,
    "device": "cuda" if torch.cuda.is_available() else "cpu",
}

model = MyModel().to(config["device"])
optimizer = torch.optim.AdamW(model.parameters(), lr=config["lr"])
criterion = nn.CrossEntropyLoss()
```

```
train_loader = DataLoader(train_dataset, batch_size=config["batch_size"], shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=config["batch_size"], shuffle=False)

# Ders 4 (list):
train_losses = []
val_losses = []
best_val_loss = float('inf')
patience_counter = 0

# DIS DONGU - epochs:
for epoch in range(config["epochs"]):
    # === EGITIM FAZI ===
    model.train()
    epoch_train_losses = []

    # IC DONGU - batch:
    for batch_idx, batch in enumerate(train_loader):
        # Ders 4 (tuple unpacking):
        images = batch["image"].to(config["device"])
        labels = batch["label"].to(config["device"])

        # Ders 5 (fonksiyon):
        logits = model(images)
        loss = criterion(logits, labels)

        # Ders 6 (if) + Ders 8 (continue):
        if torch.isnan(loss):
            print(f"NaN at epoch {epoch}, batch {batch_idx} - skip")
            continue

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        epoch_train_losses.append(loss.item())

        # Ders 6 (if) + Ders 2 (modulus):
        if batch_idx % 100 == 0:
            print(f"E{epoch} B{batch_idx}: loss={loss.item():.4f}")

    # === DOGRULAMA ===
    model.eval()
    epoch_val_losses = []

    with torch.no_grad():
        for batch in val_loader:
            images = batch["image"].to(config["device"])
```

```
            labels = batch["label"].to(config["device"])
            logits = model(images)
            loss = criterion(logits, labels)
            epoch_val_losses.append(loss.item())

    # Ders 4 (sum + len):
    avg_train_loss = sum(epoch_train_losses) / len(epoch_train_losses)
    avg_val_loss = sum(epoch_val_losses) / len(epoch_val_losses)

    train_losses.append(avg_train_loss)
    val_losses.append(avg_val_loss)

    print(f"Epoch {epoch}: train={avg_train_loss:.4f}, val={avg_val_loss:.4f}")

    # Ders 8 (break) + Ders 7 (dict):
    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        patience_counter = 0
        torch.save({
            "epoch": epoch,
            "model_state_dict": model.state_dict(),
            "val_loss": avg_val_loss,
        }, "best_model.pt")
    else:
        patience_counter += 1
        if patience_counter >= config["patience"]:
            print(f"Early stopping at epoch {epoch}")
            break

print("Egitim bitti!")
```

8.10.1 Mosh'un 8 Dersinin Tek Class'ta Sentezi

Bu **tek kod parçasında**:

Ders	Konsept	Yerleşim
1	print, sıralı yürütme	her `print(...)` ve kodun akışı
2	değişken, tip, math	`lr = 1e-3`, modulus (`batch_idx % 100`)
3	input → config dict	JSON config yükleme aynı pattern
4	list, append, tuple unpacking	`train_losses = []`, `images, labels = batch`
5	def, return	`model(images)`, `criterion(...)`
6	if, comparison	`if torch.isnan(...)`, `if avg_val_loss < best`

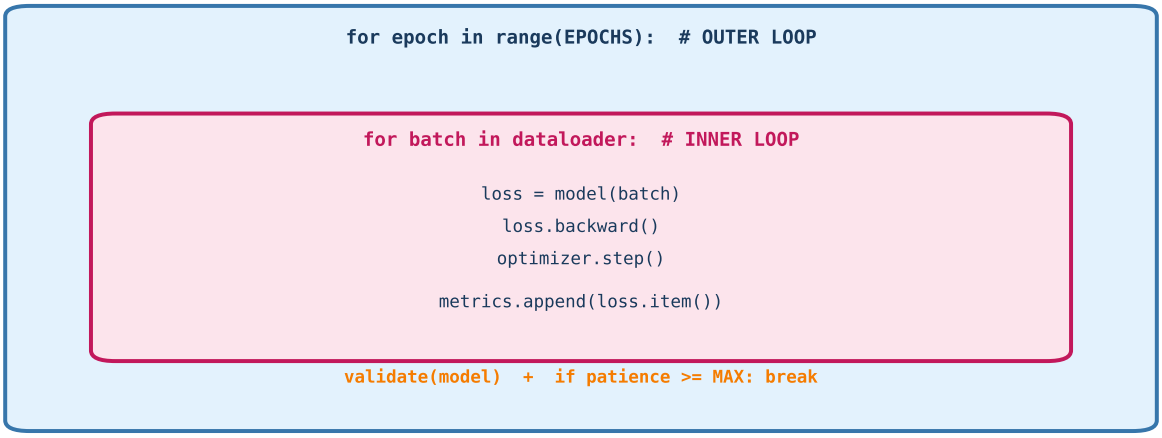
Ders	Konsept	Yerleşim
7	dict, .get	`config["lr"]`, `batch["image"]`, `model.state_dict()`
8	for, range, break, continue	tüm döngü iskeleti

Mosh'un **8 dersi**, modern AI'nın **tek bir eğitim sistemine** dökülüyor.

8.10.2 `for batch in dataloader` — Modern AI'nın Tek Cümlesi

```
for batch in dataloader:
    ...
```

Bu **dokuz karakter** modern AI'nın çekirdek mantığını taşır. GPT-4 eğitilirken bu satır milyonlarca kez çağrıldı. Stable Diffusion'da gigabayt veri akıttı.



Şekil 8.2: ML eğitim döngüsünün anatomi — outer (epochs) × inner (batches)

8.11 Bu Dersin Özeti

1. **`while kosul:`** — koşul True iken tekrar.
2. **`for x in koleksiyon:`** — her öğeye uygula.
3. **`range(start, stop, step)`** — sayı dizisi.
4. **`enumerate(...)`** — (index, eleman) tuple.
5. **`zip(a, b)`** — paralel iteration.
6. **`break`** — döngüden çık.
7. **`continue`** — bu iterasyonu atla.
8. **Nested loops** — çarpımsal.

9. **List comprehension** — `[ifade for x in iter if kosul]`.
10. **ML training loop** — Mosh'un sentezi.

> **!** Tek Bir Cümle
>
> `while` koşullu tekrar, `for` koleksiyon gezme — Mosh'un sekiz dakikalık `i <= 10` mikro örneği ve "for letter in 'Draft Academy' " pattern'ı, modern AI'nın çekirdek kodu olan `for epoch in range(EPOCHS): for batch in dataloader: train(batch)` döngüsünün **birebir atasıdır**; her sinir ağı eğitimi, her LLM fine-tune, her görüntü sınıflandırması bu döngü üzerinde inşa edilir; `break` early stopping'tir, `continue` bad batch skip'tir, `enumerate` logging'in temelidir; ve Mosh'un üç-hak Guessing Game'i, modern hyperparameter search'ün ilkel formudur.

8.12 Egzersizler

Egzersiz 1. En büyük sayıyı bul (`max()` kullanma):

```
nums = [3, 7, 1, 9, 4, 6, 8, 2]

en_buyuk = ???    # baslangic

for n in nums:
    if ???:
        en_buyuk = n

print(en_buyuk)   # 9
```

Egzersiz 2. Guessing Game (5 hak + ipucu):

```
secret = "python"
attempts = 5

for i in range(attempts):
    guess = input(f"Tahmin ({attempts - i} hak): ")
    if guess == secret:
        print("Kazandiniz!")
        break
    else:
        ipucu = ???   # eslesen harfler
        print(f"Ipucu: {ipucu}")
else:
    print(f"Kaybettiniz! Kelime: {secret}")
```

Egzersiz 3. 5×5 çarpım tablosu:

```
for i in range(1, 6):
    for j in range(1, 6):
        print(???, end=" ")
    print()
```

Egzersiz 4. (`enumerate + zip + list comp`)

```
isimler = ["Deniz", "Aylin", "Mosh"]
yaslar = [35, 28, 40]

# Klasik:
yas_dict = {}
for ad, yas in zip(isimler, yaslar):
    yas_dict[ad] = yas

# Comprehension (tek satir):
yas_dict_compact = {???}

# Enumerate ile numaralandirilmis:
sirali_dict = {???}
# {1: "Deniz", 2: "Aylin", 3: "Mosh"}
```

Egzersiz 5. (Builder eksen — fake training loop)

```
import random

dataset = [random.uniform(0, 1) for _ in range(1000)]

EPOCHS = 10
BATCH_SIZE = 50

train_losses = []
best_loss = float('inf')
patience_counter = 0
PATIENCE = 3

for epoch in range(EPOCHS):
    epoch_losses = []

    n_batches = ???   # dataset // BATCH_SIZE
    for batch_idx in range(n_batches):
        start = batch_idx * BATCH_SIZE
        end = start + BATCH_SIZE
        batch = ???   # dataset slicing

        loss = sum(batch) / len(batch) + random.uniform(-0.1, 0.1)
```

```
        if batch_idx % 5 == 0:
            print(f"E{epoch} B{batch_idx}: {loss:.4f}")

        epoch_losses.append(loss)

    avg_loss = ???
    train_losses.append(avg_loss)

    print(f"Epoch {epoch} avg: {avg_loss:.4f}")

    if avg_loss < best_loss:
        best_loss = avg_loss
        patience_counter = 0
    else:
        patience_counter += 1
        if patience_counter >= PATIENCE:
            print("Early stopping!")
            break

print(f"\nBitti! Best loss: {best_loss:.4f}")
```

PyTorch eğitim kodunun **iskeleti**ni birebir anlamış olacaksın.

8.13 Sonraki Ders İçin Hazırlık

Ders 9: Uygulama — Çevirmen ve İyi Yorum Alışkanlığı

Kısa bir pratik ders. Mosh'un Ch 25 + Ch 26 (Translator + Comments).

- **Mosh'un Ch 25-26'sını izle** (2:52:44-3:04:23, ~12 dk).
- **Şu cümleyi içselleştir:** "Yorum kodu çalışmaz ama okuyucuya rehberlik eder."

> Bu dersten tek bir şey alıp gideceksen
>
> `for batch in dataloader:` — bu dokuz karakter modern AI'nın çekirdek mantığını taşır; Mosh'un Ch 22'de öğrettiği `for letter in "Draft Academy":` ve Ch 20'de öğrettiği `while i <= 10:` mikro örnekleri, GPT-4 eğitilirken, Stable Diffusion'da, BERT fine-tune'larında **birebir** aynı yapıyla milyonlarca kez çağrıldı; `break` early stopping'dir, `continue` bad batch skip'tir, `enumerate` logging'dir, nested loops vectorization kararının kapısıdır — Mosh'un sekiz dersinde öğrettiği her şey bu derste tek bir ML eğitim sisteminde birleşiyor.

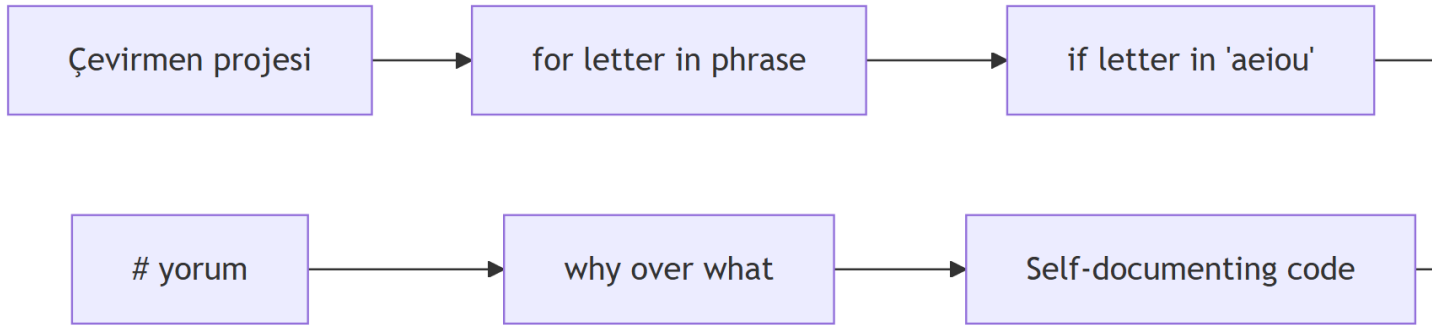
9 Uygulama: Çevirmen ve İyi Yorum Alışkanlığı

String transform + # yorum — NLP preprocessing ve ML documentation

> Bölüm bilgisi
>
> - **Mosh'un videosu:** Chapters 25-26 (Translator → Comments) (12 dk)
> - **Bölüm aralığı:** 2:52:44 — 3:04:23
> - **Kaynaklar:** PEP 8 — comments · PEP 257 — docstrings
> - **Okuma süresi:** 30 dk

9.1 Bu Derste Ne Var?

Şu ana kadar öğrendiğimiz tüm yapıları (değişken, fonksiyon, if, for) bir araya getirip **uygulama** yazma alıştırması yapacağız. Ayrıca programlama disiplininin görünmez ama kritik bir parçasını öğreneceğiz: **iyi yorum yazma alışkanlığı**.



Şekil 9.1: Ders 9'un akış haritası — çevirmen ve yorum disiplini

Dersin iki parçası:

1. **Çevirmen projesi (Mosh Ch 25).** — "Draft dili"ne çeviren küçük program. Ders 4-5-6-8'in sentezi.
2. **Yorumlar (Mosh Ch 26).** — # ile yazılan, **insanın okuması için kritik** satırlar.

Builder Notu — Bu Dersin ML Köprüleri

- **Çevirmen = preprocessing pipeline.** Karakter karakter dönüşüm = string normalization (lowercase, accent stripping), NLP'nin temeli.
- **`for letter in phrase` = tokenize/character-level encoding.** Modern transformer'larda `for token in tokens:` ile eşleşir. Mosh'un karakter döngüsü, GPT'nin BPE'sinin ilkel formu.
- **String transformation = data cleaning.** ML pipeline'ının %30'u veri temizleme: lowercase, punctuation removal, whitespace normalization.
- **Yorum = ML pipeline documentation.** Sinir ağı kodu yalnız bakıldığında anlaşılmaz: "neden 512 dim?", "neden bu loss?", "neden bu augmentation?".
- **Docstring = API documentation (PyTorch, NumPy).** `help(torch.nn.Linear)` yüzlerce satırlık docs açar.
- **"Why over what" prensibi.** İyi yorum koda **ne yaptığını** değil, **neden yaptığını** açıklar.
- **Magic number = yorum gereksinimi.** `lr = 1e-3` yalnız görülürse "neden 1e-3?" sorusu kalır.
- **Self-documenting code prensibi.** En iyi yorum **yorumsuz kod**'tur.

9.2 Çevirmen Projesi

Mosh kuruyor:

> *"tercüman nasıl kurulacağını göstereceğim piton. yani esasen yapabiliriz içeri almak onu bir cümle veya kelime gibi götürebiliriz ve onu farklı bir dile çevirebileceklerdir."* — Mosh (Türkçe dublaj), 2:52:44

9.2.1 Draft Language

"Draft Language" Mosh'un uydurduğu dil:

1. Tüm **ünlüleri** (`a`, `e`, `i`, `o`, `u`) `g` ile değiştir.
2. Sessiz harfler aynen kalsın.

Örnekler:

- `dog` → `dgg`
- `cat` → `cgt`
- `python` → `pythgn`

9.2.2 İlk Versiyon

```
def translate(phrase):
    translation = ""
    for letter in phrase:
        if letter in "aeiouAEIOU":
            translation = translation + "g"
        else:
            translation = translation + letter
    return translation

print(translate(input("Bir cumle gir: ")))
```

Çalıştır:

```
Bir cumle gir: To be or not to be
Tg bg gr ngt tg bg
```

9.2.3 Mantığın Analizi

Satır	Ders	Konsept
`def translate(phrase):`	5	Fonksiyon tanımı
`translation = ""`	4	Akumulator (boş başla)
`for letter in phrase:`	8	String iteration
`if letter in "aeiouAEIOU":`	6 + 7	Membership test
`translation += "g"`	2	String concat
`return translation`	5	Değer döndürme

Bu mikro program **Ders 4-5-6-8'in pratik sentezi.**

9.2.4 İyileştirme — `.lower()` ile Verimli

> *"Aslında sadece soyleyebilirim ki mektup nokta alt icinde ve simdi sadece kucuk harfleri yazmam gerekiyor."* — Mosh (Türkçe dublaj), 2:58:32

```
def translate(phrase):
    translation = ""
    for letter in phrase:
        if letter.lower() in "aeiou":   # case-insensitive
            translation += "g"
        else:
            translation += letter
    return translation
```

9.2.5 İyileştirme — Büyük/Küçük Harf Koruma

Şu an `"To"` → `"Tg"`. Asimetri:

```
def translate(phrase):
    translation = ""
    for letter in phrase:
        if letter.lower() in "aeiou":
            if letter.isupper():
                translation += "G"
            else:
                translation += "g"
        else:
            translation += letter
    return translation

print(translate("To Be Or Not To Be"))
# Tg Bg Gr Ngt Tg Bg
```

9.2.6 Modern Versiyon — `join` + Conditional Expression

```
def translate(phrase):
    """Bir cumleyi Draft diline cevirir (ünlüler G/g olur)."""
    sesli = "aeiou"
    parcalar = []

    for harf in phrase:
        if harf.lower() in sesli:
            yeni = "G" if harf.isupper() else "g"
        else:
            yeni = harf
        parcalar.append(yeni)

    return "".join(parcalar)
```

- **`list + .join()`** yerine string concat — büyük metinlerde daha hızlı.
- **Conditional expression** (`"G" if ... else "g"`) — Ders 6.
- **Docstring** (`"""..."""`) — Ders 5.

9.2.7 Tek Satır — List Comprehension

```
def translate(phrase):
    return "".join(
        ("G" if h.isupper() else "g") if h.lower() in "aeiou" else h
```

```
        for h in phrase
    )
```

Bir satırda her şey. Pratik öneri: **2-3 koşul iç içe OK, daha fazlasıysa açık `for`**.

9.2.8 Çevirmen = Preprocessing

Builder Notu — NLP Pipeline

```
def clean_text(text):
    """Bir metni ML-ready hale getir."""
    # 1. Lowercase
    text = text.lower()

    # 2. Punctuation strip
    import string
    text = "".join(c for c in text if c not in string.punctuation)

    # 3. Whitespace normalize
    text = " ".join(text.split())

    return text

raw = "Hello WORLD!!! Today is 2026..."
cleaned = clean_text(raw)
print(cleaned)
# hello world today is 2026
```

Mosh'un draft language çevirmeni, bu **ML preprocessing pipeline**'ının ilkel formu. **Tokenization** — bir adım sonra:

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
tokens = tokenizer.tokenize("Hello world!")
print(tokens)   # ['hello', 'world', '!']

ids = tokenizer.encode("Hello world!")
print(ids)      # [101, 7592, 2088, 999, 102]
```

Mosh'un `for letter in phrase`'ı, transformer'larda `for token in tokens`'a evrilir.

9.3 Yorumlar — # ile İnsan-Okunabilirlik

Mosh Ch 26 kısa ama temel bir kavramı işliyor:

> *"bir yorum [...] python tarafından işlenmeyecek olan bir çizgidir."* — Mosh (Türkçe dublaj), 3:00:47

9.3.1 Sentaks

```
# Bu bir yorum. Python bunu çalıştırmaz.

x = 5  # Satır içi yorum: x'i 5 yap.
```

İki form:

- **Tam satır yorum** — `#` ile başlar.
- **Satır-içi yorum** — kod sonrasında `#`.

9.3.2 Python Yorum Yok Sayar

```
# print("bu basilmaz")
print("bu basilir")   # Yorum kismi degil, bu kod basilir
```

> *"Temelde sadece insanlar için kullanılır. bu yüzden benim veya başka bir geliştiricinin beğenisine küçük bir yorum yazın"* — Mosh (Türkçe dublaj), 3:01:09

9.3.3 Çoklu Satır

```
# Yontem 1: Her satiri # ile baslat (PEP 8 önerisi)
# Bu bir uzun
# açıklama.

"""
Yontem 2: Triple-quoted string.
Etkisiz (atanmamış).
"""
```

9.3.4 Docstring vs Comment

```
def translate(phrase):
    """Cevirmen - bir cumleyi Draft diline cevirir.

    Args:
        phrase: Cevirilecek string.

    Returns:
        Cevirilmis string.
    """
    # Akumulator: gezerken doldurulacak
    translation = ""

    # Her karakter icin kontrol
    for letter in phrase:
        ...

    return translation
```

- **Docstring (`"""..."""`)** — fonksiyon başında. `help(...)` ile görünür. **Resmi**.
- **Comment (`#`)** — kod akışında. **Geçici/açıklayıcı**.

Pratik kural: Fonksiyon imzaları → docstring. Kod akışındaki kararlar/uyarılar → `#`.

9.4 İyi Yorum Alışkanlığı

9.4.1 "Why Over What" Prensibi

Kötü yorum — kod tekrarı:

```
# x'i 5 yap
x = 5

# i'yi 1 artir
i += 1

# Loss'u backward yap
loss.backward()
```

Değersiz. Kod zaten söylüyor.

İyi yorum — gerekçe:

```
# Magic number: Paper X'te kanıtlanan optimal hyperparameter
LEARNING_RATE = 3e-4

# Onceki epoch'tan kalan gradient'leri sifirla
```

```
# (PyTorch otomatik clear etmez - bilincli tasarim)
optimizer.zero_grad()

# Augmentation: %50 ihtimalle horizontal flip
# Gerekce: ImageNet'te 1% accuracy boost (validation)
if random.random() < 0.5:
    image = transforms.functional.hflip(image)
```

Bunlar değer katıyor — "neden bu sayı? Kaynak nerede?" sorularını yanıtlıyor.

9.4.2 Self-Documenting Code

```
# KOTU - yorum kodu kompanse:
x = 0   # epoch sayisi
y = 100  # max epoch
for i in range(y):
    z = train(model, data)   # z = train loss

# IYI - kod kendi acikliyor:
epoch_count = 0
MAX_EPOCHS = 100
for epoch_idx in range(MAX_EPOCHS):
    train_loss = train(model, data)
    epoch_count += 1
```

İyi isimlendirme = yorum gereksizliği.

9.4.3 Yorum Tehlikeleri

⚠ Üç tehlike

1. Yanlış / Eski Yorum:

```
# learning rate 1e-3 (KOD DEĞIŞTI, YORUM KALMADI)
LEARNING_RATE = 5e-4
```

Eski yorum yorumsuz koddan **daha kötü** — okuyucuyu yanıltır.
2. Aşırı Yorum:

```
i = 0          # i'yi sifirla
i += 1         # i'yi bir artir
i *= 2         # i'yi iki ile carp
```

Görsel akışı bozar.
3. Şaka / Sızlanma:

```
# Burayi düzenleyene lanet olsun
# Bu fonksiyon calisiyor ama nasil bilmiyorum
```

Profesyonel kodda yer almaz. TODO'lar issue tracker'a.

9.4.4 İyi Yorum Pattern'ları

Pattern 1 — Block başlığı:

```
# ===== Hyperparameters =====
LEARNING_RATE = 1e-3
BATCH_SIZE = 64

# ===== Model =====
model = MyModel()

# ===== Training =====
for epoch in range(EPOCHS):
    ...
```

Pattern 2 — Algoritma açıklaması:

```
def find_lr(model, loader):
    """Learning rate finder (Smith 2018)."""
    # Adim 1: LR'yi exponential olarak artir
    lr_schedule = [1e-7 * 10**i for i in range(50)]

    # Adim 2: Her LR icin bir batch egit
    losses = []
    for lr in lr_schedule:
        ...

    # Adim 3: Loss vs LR plotunda dik dustugu noktayi sec
    return lr_schedule[find_steepest_drop(losses)]
```

Pattern 3 — Uyarı:

```
# UYARI: Bu fonksiyon model.train() modunu cagirmaz
# Cagiran kod sorumlu (genelde validation kodu disarida)
def evaluate(model, loader):
    ...
```

Pattern 4 — Referans:

```
# Reference: Attention is All You Need (Vaswani 2017)
# Section 3.2.1, Eq. (1)
def scaled_dot_product_attention(Q, K, V):
    d_k = Q.shape[-1]
    scores = Q @ K.transpose(-2, -1) / (d_k ** 0.5)
    attn = scores.softmax(dim=-1)
    return attn @ V
```

ML araştırma kodunda paper referansı kritik.

9.4.5 ML Pipeline Documentation Hiyerarşisi

Builder Notu — Üç Katmanlı Disiplin

```
"""
training.py - Görsel sınıflandırma modeli eğitim scripti.

Yazar: Deniz
Tarih: 2026-05-28
Referans: ResNet paper (He et al. 2015)
"""

import torch

# Kaynak: configs/resnet18.yaml
config = load_config("configs/resnet18.yaml")


def train_one_epoch(model, loader, optimizer, criterion):
    """Tek bir epoch'un eğitim adımı.

    Args:
        model: PyTorch modeli (nn.Module).
        loader: DataLoader, batch'leri dict olarak yield eder.
        optimizer: AdamW veya benzeri.
        criterion: Loss fonksiyonu.

    Returns:
        avg_loss: Epoch'un ortalama loss'u.
    """
    model.train()
    losses = []

    for batch_idx, batch in enumerate(loader):
        # Veri device'a taşı
        images = batch["image"].to(DEVICE)
        labels = batch["label"].to(DEVICE)

        # Forward + loss
        logits = model(images)
        loss = criterion(logits, labels)

        # NaN guard - bozuk batch'i atla
        if torch.isnan(loss):
            continue

        # Gradient clipping (paper sec 3.4)
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        losses.append(loss.item())

    return sum(losses) / len(losses)
```

Üç katman:

1. **Module docstring** — dosya seviyesinde.
2. **Function docstring** — her fonksiyon başında.
3. **Inline comments** — kritik karar noktalarında.

9.5 Bu Dersin Özeti

1. **Çevirmen projesi** — string transform pipeline'ı, Ders 4-5-6-8 sentezi.
2. **Yorumlar (#)** — Python yok sayar, insan okur.
3. **İyi yorum: "why over what".**
4. **Self-documenting code** — iyi isimlendirme yorum azaltır.
5. **Docstring vs comment** — fonksiyon imzaları docstring, akış yorumları #.
6. **Magic number = yorum gereksinimi.**
7. **ML pipeline doc:** module → function → inline hiyerarşisi.

! Tek Bir Cümle

Mosh'un `for letter in phrase: if letter in "aeiou": translation += "g"` mikro programı, modern ML preprocessing pipeline'ının (lowercase, punctuation strip, tokenize) ve transformer karakter-bazlı encoding'in birebir atasıdır; yorumlar (#) Python tarafından yok sayılır ama kodun **neden** öyle yazıldığını insanlara taşır — magic number'lar, paper referansları, algoritma adımları; iyi yorum "ne yapar" demek değil "neden yapar" demektir, ve en iyi yorum **gereksiz** yorumdur (self-documenting code).

9.6 Egzersizler

Egzersiz 1. Türkçe ünlüler için çevirmen — `a, e, ı, i, o, ö, u, ü`:

```
def translate_turkce(phrase):
    """Bir Turkce cumleyi Draft diline cevirir."""
    turkce_unluler = "aeıioöuü"
    translation = ""

    for letter in phrase:
        if letter.lower() in turkce_unluler:
            translation += "G" if letter.isupper() else "g"
        else:
            translation += letter

    return translation

print(translate_turkce("Çok güzel bir gün"))
# Çgk ggzgl bgr ggn
```

Egzersiz 2. Sesli harf sayısı:

```
def count_vowels(phrase):
    count = 0
    ???
    return count

print(count_vowels("Hello World"))   # 3
print(count_vowels("AEIOU"))          # 5
```

Egzersiz 3. Yorumsuz kodu **iyi yorumlarla** zenginleştir:

```
import torch
import torch.nn as nn

config = {"lr": 3e-4, "batch_size": 32, "epochs": 50}

model = nn.Sequential(
    nn.Linear(784, 256),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(256, 10),
)

optimizer = torch.optim.AdamW(model.parameters(), lr=config["lr"])
criterion = nn.CrossEntropyLoss()

for epoch in range(config["epochs"]):
    for batch in train_loader:
        x, y = batch
        logits = model(x)
        loss = criterion(logits, y)
        optimizer.zero_grad()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
```

Konular: `3e-4` (Karpathy tweet?), `0.2` dropout, AdamW vs Adam, gradient clip 1.0.

Egzersiz 4. Sezar şifresi:

```
def caesar(phrase, shift):
    """Sezar sifresi - her harfi shift kadar kaydir."""
    result = ""
    for letter in phrase:
        if letter.isalpha():
            base = ord("A") if letter.isupper() else ord("a")
```

```
            offset = (ord(letter) - base + shift) % 26
            result += chr(base + offset)
        else:
            result += letter
    return result

print(caesar("Hello World", 3))    # Khoor Zruog
print(caesar("Khoor Zruog", -3))   # Hello World
```

`ord()` ve `chr()` — karakter ASCII sayı.

Egzersiz 5. (Builder eksen — NLP preprocessing)

```
def preprocess_text(text):
    """ML-ready hale getir.

    Adimlar:
    1. Lowercase
    2. Punctuation strip
    3. Whitespace normalize
    4. Strip start/end
    """
    import string

    # 1. Lowercase
    text = text.lower()

    # 2. Punctuation
    text = "".join(c for c in text if c not in string.punctuation)

    # 3. Whitespace
    text = " ".join(text.split())

    # 4. Strip
    text = text.strip()

    return text

raw = "Hello, World!!! Today is 2026... great day."
clean = preprocess_text(raw)
print(clean)
# "hello world today is 2026 great day"
```

NLP pipeline'larında kritik fonksiyon.

9.7 Sonraki Ders İçin Hazırlık

Ders 10: Hata Yönetimi ve Dosyalar

Mosh'un Ch 27-29 (Try/Except, Reading Files, Writing Files). Programlarımız artık **dış dünyaya** dokunacak.

- **Mosh'un Ch 27-29'unu izle** (3:04:23-3:28:16, ~24 dk).
- **Şu cümleyi içselleştir:** "Hata olabilir, dosya olabilir veya olmayabilir — program bu belirsizlikle başa çıkmalı."

> Bu dersten tek bir şey alıp gideceksen
>
> Mosh'un çevirmeni — `for letter in phrase: if letter in "aeiou": translation += "g"` — modern ML preprocessing pipeline'ının (NLP tokenization, character encoding, BPE) çekirdek pattern'ıdır; yorumlar (`#`) Python'ın **çalıştırmadığı** ama insanın okuması gereken kritik bilgi taşıyıcılarıdır; en iyi yorum **gereksiz** yorumdur (self-documenting code), ama gerektiğinde "ne" değil "neden" yazılır.

10 Hata Yönetimi ve Dosyalar

`try/except` + `with open` — NaN guard ve checkpoint persistence

> **Bölüm bilgisi**
>
> - **Mosh'un videosu:** Chapters 27-29 (Try/Except → Reading → Writing) (24 dk)
> - **Bölüm aralığı:** 3:04:23 — 3:28:16
> - **Kaynaklar:** Python docs — errors · Python docs — files · PEP 343 — with statement
> - **Okuma süresi:** 45 dk

10.1 Bu Derste Ne Var?

Programlarımız artık **dış dünyaya** dokunacak: kullanıcının yanlış girdisi, eksik dosya, bozuk veri, network kesintisi. Bu derste Python'un bu belirsizliklere karşı **savunmasını** öğreneceğiz.

Şekil 10.1: Ders 10'un akış haritası — try/except'ten ML defansif kodunan

Dersin beş parçası:

1. **`try/except` ile hata yakalama.**
2. **Spesifik exception türleri.** — `ValueError`, `ZeroDivisionError`, `FileNotFoundError`, `TypeError`.
3. **`with` statement — context manager.**
4. **Dosya okuma.** — `open(...).read()`, `readlines()`, line iteration.

5. **Dosya yazma.** — `"w"` mode, `"a"` append.

Builder Notu — Bu Dersin ML Köprüleri

- **`try/except` = ML training defansif kod.** Üretim ML kodu `try/except` ile sarılı. `try: loss.backward() except RuntimeError as e: handle_oom(e)` — GPU bellek aşan batch'i akanı bırakmadan toparla.
- **NaN guard pattern.** `try: loss = compute_loss(batch) except ValueError: continue`.
- **Specific exceptions = üretim kalite imzası.** Sade `except:` profesyonel kodda **yasak**.
- **`with open(...)` = context manager pattern.** ML'de `with torch.no_grad():` (inference), `with autocast():` (mixed precision), `with model.no_sync():` — hepsi aynı `with` mekanizması.
- **Dosya okuma = data loading.** `pd.read_csv`, `torch.load`, `Image.open`, `json.load` — Mosh'un `open()` türevleri.
- **Dosya yazma = checkpoint persistence.** `torch.save(model.state_dict(), "model.pt")` arka planda `open("model.pt", "wb")`.
- **Atomic write pattern.** `.tmp` + `os.replace` — ML için kritik (crash safety).
- **JSON/pickle = ML config + state.** Config dosyaları + model state'ler.
- **Logging = log file write.** `print` debug için OK; `logging.info(...)` üretim.

10.2 Hata Kavramı

Mosh:

> *"çoğu zaman bunlar [...] programınızın çalışmasını tamamen durdurmasına rağmen gerçekleşir. ve aslında yapabileceğimiz şey, dikkat edebileceğimiz."* — Mosh (Türkçe dublaj), 3:04:41

10.2.1 Hata = Exception

Python'da bir hata oluştuğunda **exception** (istisna) fırlatılır.

```
sayi = int(input("Sayi gir: "))   # kullanici "abc" girerse hata
print(10 / sayi)                  # kullanici 0 girerse hata
```

Kullanıcı `"abc"` girerse:

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

Program **çöker**.

10.2.2 Yaygın Exception Türleri

```
# ValueError - yanlis tipte ama dogru sentaksta:
int("abc")              # ValueError
float("merhaba")        # ValueError

# TypeError - tip uyumsuzlugu:
"merhaba" + 5           # TypeError
len(42)                 # TypeError

# ZeroDivisionError:
10 / 0                  # ZeroDivisionError

# KeyError - dict'te yok:
{"a": 1}["b"]           # KeyError: 'b'

# IndexError:
[1, 2, 3][100]          # IndexError

# NameError - tanimsiz:
print(undefined_var)    # NameError

# FileNotFoundError:
open("yok.txt")         # FileNotFoundError

# AttributeError:
"abc".reverse()         # AttributeError (string'in reverse'i yok)
```

Her tip **kendi anlamını taşır**.

10.2.3 ML'de Exception'lar

Builder Notu — PyTorch Exception'ları

```python
import torch

# torch.cuda.OutOfMemoryError:
try:
    x = torch.randn(10000, 10000, device="cuda")
except torch.cuda.OutOfMemoryError:
    print("GPU bellek yetersiz")

# RuntimeError - shape mismatch:
try:
    y = torch.randn(3, 4) @ torch.randn(5, 6)
except RuntimeError as e:
    print(f"Tensor shape uyumsuz: {e}")

# ValueError - model konfig hatasi:
try:
    model.load_state_dict(wrong_state_dict)
except ValueError as e:
    print(f"State dict mismatch: {e}")
```

Her tür farklı sorun → farklı çözüm. **`except:` ile hepsini yakalamak = sorunları görmezden gelmek.**

10.3 `try/except` — Hata Yakalama

Mosh:

> *"blok dışında denen bir şey bir denemede kabul et blog temelde programlarınızın denemesine izin verir"* — Mosh (Türkçe dublaj), 3:06:38

10.3.1 Sentaks

```python
try:
    # tehlikeli kod
    sayi = int(input("Sayi gir: "))
    print(10 / sayi)
except:
    # hata olursa buraya gel
    print("Bir sorun olustu!")
```

Akış:

1. try bloğu çalışmaya başlar.
2. Hata yoksa: except atlanır.
3. Hata olursa: try'ın geri kalanı atlanır, except çalışır.

10.3.2 Çıplak except: — YASAK

Mosh:

> *"Python'da en iyi yöntem bunları kullanmaktır. belirli bir hata olduğundan her zaman kabul etmek istersiniz"* — Mosh (Türkçe dublaj), 3:12:21

```
# YASAK:
try:
    risky_function()
except:
    pass    # her şeyi yutmuş
```

Bir bug ne olduğunu öğrenmek imkansızlaşır.

10.3.3 Spesifik except

```
try:
    sayi = int(input("Sayi gir: "))
    print(10 / sayi)
except ValueError:
    print("Sayi olarak gecerli degil.")
except ZeroDivisionError:
    print("Sifira bolunemez.")
```

Üç senaryo, üç farklı işlem.

10.3.4 Birden Fazla Exception Tek Blokta

```
try:
    ...
except (ValueError, TypeError, KeyError):
    print("Bilinen bir hata")
```

10.3.5 Exception'ı Yakala — as

```
try:
    sayi = int(input("Sayi gir: "))
except ValueError as e:
    print(f"Hata detayi: {e}")
    # Hata: invalid literal for int() with base 10: 'abc'
```

ML kodda kritik:

```
try:
    model.load_state_dict(state)
except RuntimeError as e:
    print(f"State dict yukleme hatasi: {e}")
    # Hangi katmanin sorunlu oldugunu söyler
```

10.3.6 `else` ve `finally`

```
try:
    sayi = int(input("Sayi: "))
except ValueError:
    print("Gecersiz!")
else:
    # try BASARIYLA bittiyse:
    print(f"Basarili: {sayi}")
finally:
    # HER DURUMDA:
    print("Temizlik kodu burada")
```

10.3.7 ML Production Hata Yönetimi

Builder Notu — Defansif ML Kodu

```python
import torch
import logging

logger = logging.getLogger(__name__)

def safe_train_step(model, batch, optimizer):
    """Tek bir egitim adimi, defansif sarmalanmis."""
    try:
        loss = model(batch).loss

        # NaN guard:
        if torch.isnan(loss) or torch.isinf(loss):
            logger.warning(f"Invalid loss: {loss.item()}, skipping batch")
            return None

        optimizer.zero_grad()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        optimizer.step()
        return loss.item()

    except torch.cuda.OutOfMemoryError as e:
        logger.error(f"OOM: {e}, clearing cache")
        torch.cuda.empty_cache()
        return None

    except RuntimeError as e:
        logger.error(f"PyTorch runtime error: {e}")
        return None

    except Exception as e:
        logger.exception(f"Unexpected: {e}")
        raise   # bilinmeyen hatalari yukseltel
```

ML üretim disiplini:

1. Spesifik exception'lar — her biri ayrı.
2. Logging — `print` değil, `logger`.
3. OOM toparlama — cache clear, batch atla.
4. Bilinmeyen hataları yükselt — silent fail YASAK.

10.4 `with` Statement — Context Manager

Modern Python'un en güzel deyimlerinden.

10.4.1 Sorun

```
file = open("data.txt", "r")
content = file.read()
file.close()
```

Eğer `file.read()` hata atarsa, `file.close()` çalışmaz. Dosya **açık kalır** (resource leak).

10.4.2 `with` Çözümü

```
with open("data.txt", "r") as file:
    content = file.read()

# file otomatik kapatildi
```

`with` Python'un sözünü tuttuğu syntax: **bloktan çıkınca otomatik temizlik** garanti. Hata olsa bile.

10.4.3 `with` ML'in Her Yerinde

Builder Notu — Context Manager Pattern

```python
import torch

# 1. Gradient hesabini gecici kapat (inference):
with torch.no_grad():
    predictions = model(test_data)

# 2. Mixed precision (FP16):
from torch.cuda.amp import autocast
with autocast():
    output = model(input)

# 3. Distributed training - gradient sync gecici kapat:
with model.no_sync():
    loss.backward()

# 4. Custom context manager - evaluation mode:
from contextlib import contextmanager

@contextmanager
def evaluation_mode(model):
    model.eval()
    try:
        with torch.no_grad():
            yield model
    finally:
        model.train()

with evaluation_mode(model):
    eval_loss = validate(model, val_loader)
# model otomatik train() moduna dondu
```

PyTorch'ta `with` **her yerde**.

10.5 Dosya Okuma

Mosh:

> *"bilgi almak için bu dosyaları kullanabilirsiniz."* — Mosh (Türkçe dublaj), 3:13:10

10.5.1 `open(...)` Modları

```
file = open("data.txt", "r")
```

Mode	Yapar	Dosya yoksa	Dosya varsa
`"r"`	Oku	Hata	Aç
`"w"`	Yaz	Yarat	**Sıfırla**
`"a"`	Append	Yarat	Aç, sona ekle
`"x"`	Exclusive	Yarat	Hata
`"rb"`	Binary oku	Hata	Aç
`"wb"`	Binary yaz	Yarat	Sıfırla

10.5.2 Üç Okuma Methodu

```
with open("data.txt") as f:
    # Yontem 1: tum dosya tek string
    content = f.read()

with open("data.txt") as f:
    # Yontem 2: satir satir liste
    lines = f.readlines()
    # ['ilk satir\n', 'ikinci satir\n', ...]

# Yontem 3: satir satir iteration (EN VERIMLI)
with open("data.txt") as f:
    for line in f:
        print(line.strip())
```

`for line in f:` Python'un en sevdiği deyim. Bellek-verimli — büyük dosyalarda tüm dosyayı belleğe almaz.

10.5.3 Defansif Pattern

```
try:
    with open("config.json") as f:
        config = json.load(f)
except FileNotFoundError:
    print("Config yok, default'lara dusulecek")
    config = {"lr": 1e-3, "batch_size": 32}
```

10.5.4 Dosya Okuma = ML Veri Yükleme

Builder Notu — Her Şey open()'ın türevi

```
# 1. CSV:
import pandas as pd
df = pd.read_csv("train.csv")    # arka planda open(..., "r")

# 2. JSON:
import json
with open("config.json") as f:
    config = json.load(f)

# 3. Pickle:
import pickle
with open("model.pkl", "rb") as f:    # binary read!
    model = pickle.load(f)

# 4. PyTorch model:
import torch
state = torch.load("model.pt")    # arka planda open(..., "rb") + pickle

# 5. Image:
from PIL import Image
img = Image.open("cat.jpg")

# 6. Text corpus (NLP):
with open("corpus.txt", encoding="utf-8") as f:
    for line in f:
        process_sentence(line.strip())

# 7. Hugging Face dataset:
from datasets import load_dataset
dataset = load_dataset("imdb")    # arka planda yuzlerce dosya okur
```

Hepsi Mosh'un `open(...)` kavramının türevleri.

10.6 Dosya Yazma

10.6.1 Write Mode `"w"` — Sıfırlar

```
with open("output.txt", "w") as f:
    f.write("Ilk satir\n")
    f.write("Ikinci satir\n")
```

⚠ DİKKAT

`"w"` modu dosyayı **siler** ve yeniden yazar. Eski içerik **gider**.

10.6.2 Append Mode `"a"` — Sona Ekler

```
with open("log.txt", "a") as f:
    f.write("Yeni log entry\n")
```

`"a"` mevcut dosyaya **eklenir**. Eski içerik **korunur**. Log dosyaları için ideal.

10.6.3 `print` ile Dosyaya Yazma

```
with open("output.txt", "w") as f:
    print("Birinci", file=f)
    print("Ikinci", file=f)
```

`print(..., file=f)` ile `print`'in normal davranışını dosyaya yönlendir.

10.6.4 Atomic Write Pattern

❗ Builder Notu — Crash Safety

```
# YANLIS:
torch.save(model.state_dict(), "model.pt")
# Eger crash olursa, model.pt yarim yazilmis olur (BOZUK!)

# DOGRU - atomic write:
import os
torch.save(model.state_dict(), "model.pt.tmp")
os.replace("model.pt.tmp", "model.pt")    # atomik rename
```

ML üretim kodunda her dosya yazma atomic olmalı. Power outage veya crash → bozuk dosya = tüm eğitim kaybı.

10.7 Bu Dersin Özeti

1. **`try/except`** — hata yakalama. Spesifik exception'lar.
2. **`as e`** — exception objesini değişkene al.
3. **`else`** ve **`finally`** — başarı sonrası ve her durumda.

4. **`with` statement** — context manager, otomatik temizlik.
5. **`open(path, mode)`** — dosya açma. `r`, `w`, `a`, `b`.
6. **`for line in f`** — bellek-verimli satır iteration.
7. **Atomic write** — `.tmp` + `os.replace`. ML için kritik.

> **!** Tek Bir Cümle
>
> `try/except` Python'a hata yönetimi yetisi kazandırır — Mosh'un `try: int(input(...)) except ValueError: ...` mikro örneği, modern ML kodunun NaN guard, OOM recovery, ve gradient clipping gibi defansif patternlarının atasıdır; `with open(...) as f:` context manager pattern'ı, PyTorch'un `with torch.no_grad():` ve `with autocast():` deyimlerinin birebir aynı yapısıdır; dosya okuma ML veri yüklemenin temeli (`pd.read_csv`, `torch.load`, `Image.open` hepsi `open()`'in türevidir), dosya yazma ise her epoch'tan sonra checkpoint persistence'ının temelidir.

10.8 Egzersizler

Egzersiz 1. Calculator defansif:

```
try:
    num1 = float(input("Birinci sayi: "))
    op = input("Islem (+, -, *, /): ")
    num2 = float(input("Ikinci sayi: "))

    if op == "/":
        sonuc = num1 / num2   # ZeroDivisionError olabilir
    ...
except ValueError:
    ???
except ZeroDivisionError:
    ???
```

Egzersiz 2. Boş olmayan satırların toplam karakter sayısı:

```
def total_char_count(path):
    try:
        with open(path, encoding="utf-8") as f:
            ???
    except FileNotFoundError:
        ???
```

Egzersiz 3. Append mode log fonksiyonu:

```python
from datetime import datetime

def log(message, path="app.log"):
    with open(path, "a") as f:
        f.write(f"{datetime.now().isoformat()} - {message}\n")

log("Program baslatildi")
log("Bir hata olustu")
```

Egzersiz 4. JSON config + default fallback:

```python
import json

def load_config(path, defaults):
    try:
        with open(path, encoding="utf-8") as file:
            return json.load(file)
    except FileNotFoundError:
        print(f"Config yok: {path}")
        return defaults
    except json.JSONDecodeError as e:
        print(f"Config bozuk: {e}")
        return defaults
```

Egzersiz 5. (Builder eksen — atomic checkpoint)

```python
import os
import json

def save_checkpoint(state, path):
    tmp_path = path + ".tmp"

    try:
        with open(tmp_path, "w") as f:
            json.dump(state, f)
        os.replace(tmp_path, path)    # atomik
    except Exception as e:
        print(f"Save failed: {e}")
        if os.path.exists(tmp_path):
            os.remove(tmp_path)
        raise

state = {"epoch": 5, "loss": 0.234, "params": [1, 2, 3, 4]}
save_checkpoint(state, "model.json")
```

PyTorch `torch.save` arka planda aynı şeyi yapar (pickle ile binary).

10.9 Sonraki Ders İçin Hazırlık

Ders 11: Modüller, Paketler, Pip

Mosh'un Ch 30 + Ch 35 (Modules + Pip + Interpreter). Şimdiye kadar tek dosyada yazdığımız kodu **organize** etmeyi ve **kütüphane evrenine** açılmayı öğreneceğiz.

- **Mosh'un Ch 30 ve Ch 35'ini izle** (3:28:16-3:43:58 + 4:20:46-4:26:47, ~22 dk).
- **Şu cümleyi içselleştir:** "Modüller kodu organize eder; pip dış dünya kütüphanelerine kapı açar."

> Bu dersten tek bir şey alıp gideceksen
>
> `try/except` Python'a hata yönetimi yetisi kazandırır — Mosh'un mikro `try: int(input(...)) except ValueError:` örneği, modern ML kodunun NaN guard, OOM recovery, atomic checkpoint write disiplinlerinin atasıdır; `with open(...) as f:` context manager pattern'ı, PyTorch'taki `with torch.no_grad():` ve `with autocast():` deyimlerinin **birebir** aynı yapısıdır; üretim kalite ML kodu = Mosh'un üç chapter'ında öğrettiği bu üç kavramın disiplinli uygulanışıdır.

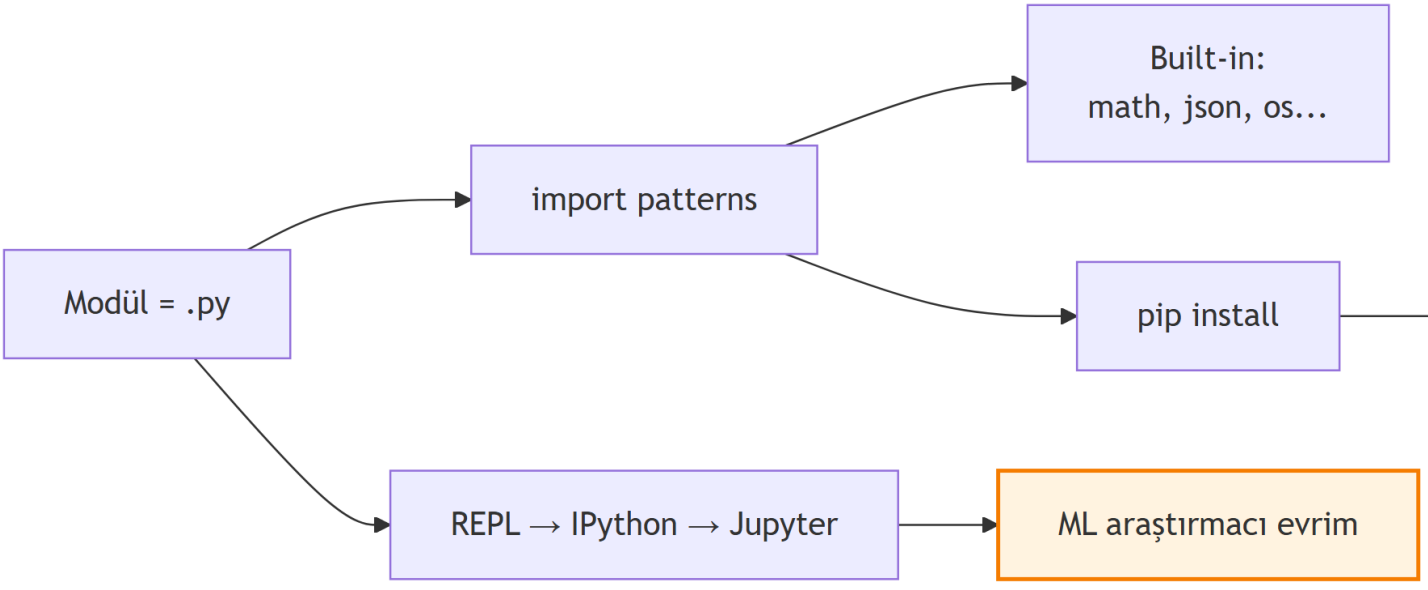
11 Modüller, Paketler, Pip

`import` + `pip install` — ML kütüphane evrenine kapı

> **Bölüm bilgisi**
>
> - **Mosh'un videosu:** Chapters 30 + 35 (Modules & Pip + Python Interpreter) (22 dk)
> - **Bölüm aralığı:** 3:28:16 — 3:43:58 + 4:20:46 — 4:26:47
> - **Kaynaklar:** Python docs — modules · PyPI · PEP 328 — imports
> - **Okuma süresi:** 45 dk

11.1 Bu Derste Ne Var?

Bu derste **kodu organize** etmeyi ve **kütüphane evrenine** açılmayı öğreneceğiz. Şimdiye kadar tek dosyada yazdık; gerçek dünyada Python kodu **modüller** halinde organize edilir, **paketler** halinde yükellenir, ve **pip** ile yüz binlerce dış kütüphane bu sisteme bağlanır.



Şekil 11.1: Ders 11'in akış haritası — modülden ML kütüphane evrenine

Dersin beş parçası:

1. **Modül nedir?** — Tek bir `.py` dosyası.
2. **`import` pattern'ları.** — `import math`, `from math import sqrt`, `import numpy as np`.
3. **Yerleşik (built-in) modüller.** — `math`, `os`, `sys`, `json`, `datetime`.
4. **`pip` ve dış paketler.** — `pip install numpy`. **ML evrenine kapı.**
5. **REPL (Mosh Ch 35).** — `python` komutuyla interaktif kabuk.

Builder Notu — Bu Dersin ML Köprüleri (Ders 1'in kapanış vaadi!)

- **`pip install` = ML kütüphane evrenine kapı.** Mosh'un Ders 1'de "Python = ML lingua franca" dediğimiz şey buradan gerçekleşir: `pip install numpy pandas torch transformers sklearn matplotlib jupyter` ile binlerce ML mühendisinin yazdığı milyonlarca satır kod bir saniyede sistemine gelir.
- **PyPI = ML topluluk çatısı.** Python Package Index (pypi.org) ~500K+ paket.
- **`import torch as t` ya da `import numpy as np`** = ML alias standartları. Kısa, evrensel: `np`, `pd`, `plt`, `torch`, `tf`, `jnp`.
- **`from X import Y` = seçimli kütüphane kullanımı.** `from torch.nn import Linear, ReLU`.
- **`venv` / `conda` / `uv`** = proje izolasyonu.
- **`requirements.txt` / `pyproject.toml`** = dependency reproducibility.
- **REPL = Jupyter notebook'un atası.**
- **`pip install -e .`** = editable install.
- **`__name__ == "__main__"`** = script vs modül ayırımı.

11.2 Modül Nedir?

Mosh:

> *"bir modül aslında sadece yapabileceğimiz bir python dosyasıdır. içine ithalat Geçerli python dosyası"* — Mosh (Türkçe dublaj), 3:28:16

Modül = bir Python dosyası (.py). İçindeki fonksiyon, değişken ve class'lar başka dosyadan **import edilebilir**.

11.2.1 Neden Modül?

1. **Kod organizasyonu.** 1000 satırlık tek dosya yerine, 10 modülden oluşan küçük dosyalar.
2. **Yeniden kullanılabilirlik.** Bir kez yaz, başka projelerinde import et.
3. **Topluluk erişimi.** PyPI'a (göreceğiz) — ML için zorunlu.

11.2.2 Mosh'un Örneği — `useful_tools.py`

```
# useful_tools.py
feet_in_mile = 5280
meters_in_kilometer = 1000

beatles = ["John", "Paul", "George", "Ringo"]

def get_file_extension(filename):
    """Bir dosya isminden uzanti dondurur."""
    return filename.split(".")[-1]

def roll_dice(num_sides):
    """num_sides yuzlu bir zar."""
    import random
    return random.randint(1, num_sides)
```

Başka dosyadan kullanım:

```
# app.py
import useful_tools

print(useful_tools.feet_in_mile)              # 5280
print(useful_tools.beatles)                    # ['John', 'Paul', 'George', 'Ringo']
print(useful_tools.get_file_extension("file.txt"))   # txt
print(useful_tools.roll_dice(10))             # 1-10 arasi
```

Mosh:

> *"`useful_tools.nokta` ve şimdi aslında erişebiliyorum Tüm bu nitelikler"* — Mosh (Türkçe dublaj), 3:31:00

`module.attribute` syntax — noktayla erişim.

11.2.3 Avantaj — Kopya-Yapıştır Yok

`yarattığın yer fark edeceksiniz Zorunda değildim herhangi bir işlevi veya değişkeni kopyala`

`useful_tools.py`'ı bir kez yazdın; şimdi 10 farklı projede import edebilirsin.

11.2.4 ML Projesi Yapısı

Builder Notu — Modern ML Projesi

ML kodu modüller halinde:

```
my_ml_project/
   config.py          # Hyperparameter'lar
   data.py            # Dataset, DataLoader
   model.py           # nn.Module sınıfları
   train.py           # Eğitim döngüsü
   evaluate.py        # Validation
   utils.py           # Yardımcı fonksiyonlar
   main.py            # Giriş noktası
```

`main.py` diğerleri import eder:

```python
from config import CONFIG
from data import build_loaders
from model import build_model
from train import train
from evaluate import evaluate

if __name__ == "__main__":
    train_loader, val_loader = build_loaders(CONFIG)
    model = build_model(CONFIG)
    train(model, train_loader, val_loader, CONFIG)
    evaluate(model, val_loader)
```

Mosh'un `useful_tools.py` mikro örneği = profesyonel ML projesinin minyatür modeli.

11.3 `import` Pattern'ları

11.3.1 `import module`

```python
import math

math.sqrt(16)     # 4.0
math.pi           # 3.141592653589793
```

Avantaj: namespace temiz, hangi modülden geldiği belli.

11.3.2 `from module import name`

```
from math import sqrt, pi, log

sqrt(16)    # math. prefix yok
```

Avantaj: kısa kod. **Dezavantaj:** namespace pollution riski.

11.3.3 `from module import *` — YASAK

```
from math import *    # YASAK

sqrt(16)   # math. sqrt mu yoksa baskasi mi?
```

Profesyonel kodda yer almaz.

11.3.4 `import module as alias`

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

ML topluluk standartları:

Modül	Standart Alias
numpy	`np`
pandas	`pd`
matplotlib.pyplot	`plt`
seaborn	`sns`
tensorflow	`tf`
torch	(yok, direkt `torch`)
jax.numpy	`jnp`

11.3.5 Hangi Pattern Ne Zaman?

```
# Genel kural - %90:
import numpy as np            # ML standardı
np.array(...)
```

```
# Eğer 1-2 isim:
from math import sqrt, pi
sqrt(16)

# REPL veya kısa script (üretimde YASAK):
from math import *
sqrt(16)
```

11.4 Built-in Modüller

Python kurulumu ~200 modülle gelir. `pip install` gerekmez.

```
# Matematik:
import math
math.sqrt(16)        # 4.0
math.pi              # 3.14159...
math.log(10)
math.factorial(5)    # 120

# Rastgele:
import random
random.random()             # 0.0 - 1.0
random.randint(1, 100)
random.choice([1, 2, 3])
random.shuffle(liste)
random.seed(42)              # reproducibility!

# Tarih/zaman:
from datetime import datetime, timedelta
now = datetime.now()
print(now.strftime("%Y-%m-%d"))

# OS:
import os
os.getcwd()
os.listdir(".")
os.path.exists("file.txt")
os.environ.get("PATH")

# JSON:
import json
data = {"ad": "Deniz", "yas": 35}
s = json.dumps(data)
loaded = json.loads(s)

with open("data.json", "w") as f:
```

```
    json.dump(data, f, indent=2)

# Collections (Ders 7'den):
from collections import Counter, defaultdict, namedtuple
c = Counter([1, 2, 2, 3, 3, 3])

# Sys:
import sys
sys.argv
sys.path

# Pathlib (modern dosya yolu):
from pathlib import Path
p = Path("data") / "train" / "img.jpg"
p.exists()
p.suffix      # .jpg
```

11.4.1 ML için Kritik Built-in'ler

Builder Notu — Günlük Kullanım

```python
# 1. Datetime - logging:
from datetime import datetime
print(f"[{datetime.now().isoformat()}] Training started")

# 2. JSON - config:
import json
with open("config.json") as f:
    config = json.load(f)

# 3. Pathlib - data paths:
from pathlib import Path
data_dir = Path("data") / "train"
images = list(data_dir.glob("*.jpg"))

# 4. Collections - class distribution:
from collections import Counter
print(Counter(labels).most_common())

# 5. Random - data shuffling + reproducibility:
import random
random.seed(42)
random.shuffle(train_data)

# 6. Math - LR schedules:
import math
lr = base_lr * math.cos(epoch * math.pi / total_epochs)
```

11.5 `pip` — Dış Paketler ve ML Evrenine Kapı

11.5.1 PyPI

pypi.org — Python resmi paket deposu. ~**500,000+ paket**.

11.5.2 `pip install`

```
pip install numpy
pip install pandas
pip install torch
pip install transformers

# Tek seferde:
```

```
pip install numpy pandas torch matplotlib

# Belirli sürüm:
pip install "numpy>=1.24,<2.0"
pip install "torch==2.0.1"
```

11.5.3 pip list ve pip show

```
pip list
pip show numpy
```

11.5.4 Virtual Environment (proje izolasyonu)

```
python -m venv .venv                # sanal ortam yarat
source .venv/bin/activate           # Linux/Mac
.venv\Scripts\activate              # Windows

pip install numpy                   # SADECE .venv'e
```

Önerilen pratik: her projeye `.venv`.

11.5.5 requirements.txt

```
numpy>=1.24
pandas>=2.0
torch>=2.0
transformers>=4.30
```

```
pip install -r requirements.txt
```

Reproducibility.

11.5.6 Modern Alternatifler

- **pyproject.toml** (PEP 518) — Python'un yeni standardı.
- **uv** (2024+) — Rust ile yazılmış, pip'ten 10-100× hızlı.
- **conda** — Anaconda ekosistemi.

11.5.7 ML Stack Setup

```
# Adim 1: Yeni proje
mkdir my_ml_project
cd my_ml_project

# Adim 2: Sanal ortam
python -m venv .venv
source .venv/bin/activate

# Adim 3: Temel ML stack:
pip install numpy pandas matplotlib scikit-learn jupyter

# Adim 4: Deep learning:
pip install torch torchvision

# Adim 5: LLM:
pip install transformers datasets accelerate

# Adim 6: Bağımlılık kaydet
pip freeze > requirements.txt
```

11.5.8 ML Kütüphane Evreni Haritası

Builder Notu — `pip install` Sonrası Açılan Evren

```
# === Temel Bilimsel ===
import numpy as np            # array, linear algebra
import scipy                  # istatistik

# === Veri ===
import pandas as pd           # DataFrame
import polars                 # pandas alternative (Rust, hızlı)

# === Klasik ML ===
import sklearn                # random forest, SVM
import xgboost                # gradient boosting

# === Derin Öğrenme ===
import torch                  # PyTorch
import torch.nn as nn
import tensorflow as tf
import jax
import jax.numpy as jnp

# === NLP / LLM ===
from transformers import AutoModel, AutoTokenizer
import langchain
import openai
import anthropic

# === Computer Vision ===
import torchvision
import PIL
import cv2                    # OpenCV

# === Görselleştirme ===
import matplotlib.pyplot as plt
import seaborn as sns

# === Notebook ===
import jupyter
import ipython

# === Production ===
import fastapi
import wandb                  # experiment tracking
import mlflow                 # ML lifecycle
```

Yukarıdaki her satır = **binlerce saat yazılmış kod**. `pip install` ile saniyede senin elinde.

Şekil 11.2: `pip install` ile açılan ML evrenı — Mosh'un `import useful_tools` mikro örneğinden binlerce paket

11.6 REPL — `python` ile İnteraktif Kabuk

Mosh'un son chapter'ı, kursun finalinde kısa bir konsept.

11.6.1 REPL Nedir?

REPL = **R**ead-**E**val-**P**rint-**L**oop.

```
$ python
Python 3.13.0 (main, ...)
>>>
```

```
>>> 2 + 2
4
>>> x = 10
>>> x * 5
50
>>> [i**2 for i in range(5)]
[0, 1, 4, 9, 16]
>>> import math
>>> math.sqrt(144)
12.0
>>> exit()
```

11.6.2 Niye Kullanılır?

1. **Hızlı deneme** — bir fonksiyonun nasıl çalıştığını dene.
2. `help(...)` — docs aç.
3. **Hesap makinesi** — `math.sqrt(2)`, `2**10`.
4. **API keşfi** — `dir(obj)`.

11.6.3 IPython — Enhanced REPL

```
pip install ipython
ipython
```

```
In [1]: x = 5
In [2]: x?           # docs gösterir
In [3]: %timeit sum(range(1000))    # benchmarking
In [4]: !ls          # shell komutu
```

ML pratiğinin standardı.

11.6.4 Jupyter Notebook

```
pip install jupyter
jupyter notebook
```

Tarayıcıda **hücre-bazlı** REPL. Markdown + kod + görselleştirme. **Modern ML araştırmasının %70-80'i** Jupyter'da yapılır.

11.6.5 REPL → IPython → Jupyter → VS Code

Builder Notu — Evrim Yolu

1. **`python` REPL** — başlangıç.
2. **IPython** — günlük (renkli, magic commands).
3. **Jupyter Notebook** — eğitim, prototip.
4. **JupyterLab** — Jupyter gelişmiş.
5. **VS Code Notebooks** — modern alternatif (Git-friendly).
6. **Google Colab / Kaggle** — bulut Jupyter, ücretsiz GPU.

ML araştırmacısı için Jupyter standardır. Production kodu `.py`'de, **deney** notebook'ta.

11.7 Bu Dersin Özeti

1. **Modül = `.py` dosyası.** Import edilebilir.
2. **`import X`, `from X import Y`, `import X as Z`** — üç ana pattern.
3. **Built-in modüller** — Python kurulumuyla gelir.
4. **`pip install`** — PyPI'dan dış paket.
5. **`venv`** — proje izolasyonu.
6. **`requirements.txt`** — reproducibility.
7. **REPL** — interaktif kabuk.
8. **IPython, Jupyter** — REPL'in evrimi.

! Tek Bir Cümle

Modül (`.py` dosyası) Python'un kodu organize etme birimi — Mosh'un `useful_tools.py` mikro örneği, modern ML projelerinin `config.py + data.py + model.py + train.py` yapısının atasıdır; **`pip install` Python ekosisteminin asıl gücüdür** — PyPI'daki 500K+ paket (numpy, pandas, torch, transformers, langchain) bir komutla sisteme gelir, ve bu Ders 1'de "Python = ML lingua franca" diye vurguladığımız şeyin **gerçek olduğu yer**; sanal ortam (`venv`) ve `requirements.txt` ML deneyimlerinin reproducibility'sini garanti eder; REPL → IPython → Jupyter hattı, Mosh'un başlangıç noktasından modern ML araştırmacısının günlük aracına uzanan evrim yoludur.

11.8 Egzersizler

Egzersiz 1. Kendi `useful_tools.py`'yi yaz:

```
# useful_tools.py
PI = 3.14159
GUNLER = ["Pzt", "Sal", "Car", "Per", "Cum", "Cmt", "Paz"]

def cember_alani(yaricap):
    return PI * yaricap ** 2

def gun_adi(idx):
    return GUNLER[idx]
```

```
# main.py
import useful_tools
print(useful_tools.cember_alani(5))    # 78.53975
print(useful_tools.gun_adi(2))         # Car
```

Egzersiz 2. Üç import pattern'i:

```python
# (a) import random
import random
print(random.randint(1, 100))

# (b) from random import randint, choice
from random import randint, choice
print(randint(1, 10))
print(choice([1, 2, 3]))

# (c) import random as r
import random as r
print(r.random())
```

Egzersiz 3. Python docs'tan **5 yeni built-in modül** keşfet (`bisect`, `heapq`, `statistics`, `functools`, `itertools`). Her biri ne işe yarar?

Egzersiz 4. Sanal ortam:

```
python -m venv .venv
.venv\Scripts\activate    # Windows
source .venv/bin/activate # Linux/Mac

pip install numpy pandas matplotlib
pip freeze > requirements.txt

cat requirements.txt
```

`numpy` bağımlılık olarak ne çağrılıyor?

Egzersiz 5. (Builder eksen — ilk ML import) REPL'de çalıştır:

```python
import numpy as np
import torch

x_np = np.array([1, 2, 3, 4, 5])
x_torch = torch.tensor([1, 2, 3, 4, 5])

print(f"NumPy: {x_np}")
print(f"Tensor: {x_torch}")

# Mosh'un Ders 2'sini tensor üzerinde:
print(f"x + 10 = {x_torch + 10}")
print(f"x ** 2 = {x_torch ** 2}")

# Mosh'un Ders 4'ünü:
print(f"x[1:4] = {x_torch[1:4]}")
print(f"len = {len(x_torch)}")
```

Mosh'un öğrettiği tüm pattern'ların — `+`, `**`, `[1:4]`, `len` — PyTorch tensor'da birebir çalıştığı.

11.9 Sonraki Ders İçin Hazırlık

Ders 12: Nesne Yönelimli Programlama (OOP) — FİNAL

Mosh'un dört chapter'ı (Classes, Quiz, Methods, Inheritance). **Python'un en güçlü** kavramı: **class**. Modern AI'nın yapı taşı.

- **Mosh'un Ch 31-34'ünü izle** (3:43:58-4:20:46, ~37 dk).
- **Şu cümleyi içselleştir:** "Class = veri + davranış paketi; bir kalıp, ondan birden fazla instance üretebilirsin."

Bu dersten tek bir şey alıp gideceksen

`pip install` Python'un asıl gücüdür — PyPI'daki 500K+ paket bir komutla sisteme gelir, ve Ders 1'de "Python = ML lingua franca" diye vurguladığımız şeyin **gerçek olduğu yer**; modüller (`.py`) ile kodu organize edersin, paketler (`__init__.py` ile klasörler) ile organizasyonu hiyerarşik hale getirirsin, `venv` ile her projeyi izole edersin; Mosh'un öğrettiği `import` mikro mekanizması, modern ML araştırmacısının her gün yaptığı `import torch.nn as nn` deyimi ile **birebir** aynı yapıdır; **bir sonraki ders (OOP) bu evrenin yapı taşlarını oluşturan `class` kavramını öğretecek**, böylece `class MyModel(nn.Module):` deyimi anlamlı olacak.

12 Nesne Yönelimli Programlama (OOP)

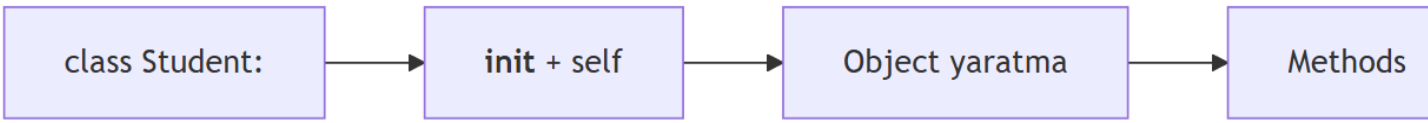
`class MyModel(nn.Module):` — modern AI'nın DNA'sı

> **Bölüm bilgisi**
>
> - **Mosh'un videosu:** Chapters 31-34 (Classes → Quiz → Methods → Inheritance) (37 dk)
> - **Bölüm aralığı:** 3:43:58 — 4:20:46
> - **Kaynaklar:** Python docs — classes · PyTorch nn.Module · PEP 8 — class names
> - **Okuma süresi:** 60 dk

12.1 Bu Derste Ne Var? — KURSUN FİNALİ

Bu kursun **son** dersi. Mosh'un başlangıçtan beri öğrettiklerini bir araya toplayıp **modern Python'un** en güçlü kavramına ulaşıyoruz: **nesne yönelimli programlama (OOP)** ve **class'lar**.

Bu, aynı zamanda **modern AI'nın yapı taşlarının** oturduğu yerdir. PyTorch'taki `class MyModel(nn.Module):` deyimi — GPT-4'ün, Stable Diffusion'ın, Claude'un, BERT'in tümünün çekirdeği — tam bu derste öğreneceğimiz kavramın uygulamasıdır.



Şekil 12.1: Ders 12'nin akış haritası — class'tan nn.Module'e ve kurs sonuna

Dersin yedi parçası:

1. **Class motivasyonu.** — Gerçek dünyayı modellemek.
2. **`class` + `__init__` + `self`.**
3. **Object yaratma + attribute erişim.**
4. **Methods.**
5. **Inheritance.**
6. **Special methods (dunders).**
7. **`nn.Module` — TAM ML sınıfı.**

Builder Notu — KURSUN FİNALİ (en derin köprüler)

- `class MyModel(nn.Module):` = **modern AI'nın DNA'sı.** Her sinir ağı, her LLM, her görüntü sınıflandırma bu pattern üzerine kurulu.
- `def __init__(self, ...):` = **layer setup.** PyTorch'ta `self.layer1 = nn.Linear(...)`.
- `def forward(self, x):` = **computation graph.** Modelin verisini girdi-çıktı zincirine dönüştürür.
- **Inheritance:** `class MyModel(nn.Module):` = **AI mimari mirası.** `nn.Module`'ün tüm özelliklerini al.
- `self` = **state encapsulation.** Mosh'un "object'in kendi referansı" PyTorch'ta `self.layers` ile model içi state.
- `super().__init__()` = **parent constructor.** PyTorch'ta **zorunlu**.
- **Special methods (dunders) = Python'un gücü.** `__call__` → `model(x)`. `__len__` → `len(dataset)`. `__getitem__` → `dataset[i]`. PyTorch DataLoader API'si bu dunders üzerine.

12.2 Class Motivasyonu

Mosh:

> *"çok var Bir dize ya da diziyi otomatikleştirmede gerçek dünyada temsil edemediğimiz şeyleri"* — Mosh (Türkçe dublaj), 3:44:53

12.2.1 Python'un Sınırlı Tipleri

Şimdiye kadar:

- `int`, `float`, `str`, `bool`, `list`, `tuple`, `dict`, `set`.

Bunlar **soyut**. Gerçek dünyada:

- **Öğrenci** — ad, yaş, GPA, bölüm, davranışlar (kayıt ol, mezun ol).
- **Ürün** — ad, fiyat, stok.
- **Sinir ağı** — katmanlar, parametreler, forward davranışı, training/eval modu.

Tek bir `int` veya `string` ile temsil edemeyiz.

12.2.2 Çözüm — Kendi Veri Tipi

> *"sınıflarla ve nesnelerle neler yapabileceğimizi temelde kendimizinkini yaratabiliriz veri tipleri."* — Mosh (Türkçe dublaj), 3:45:15

Class ile **kendi veri tipini** tanımlarsın:

```
class Student:
    pass    # geçici - boş class
```

Şimdi `Student` Python'un **yeni bir tipi**.

12.2.3 Class vs Object

> *"a class is just like an overview of what the student data type is bir obje is an actual student"* — Mosh (Türkçe dublaj), 3:51:05

- **Class** = blueprint, kalıp.
- **Object** (instance) = somut bir örnek.

Analoji:

- Class `Student` = "öğrencinin ne olduğunu tanımlayan teorik kalıp".
- Object `student1 = Student("Jim", ...)` = gerçek bir öğrenci.

12.2.4 Class ML'de

Builder Notu — Blueprint → Object

```
import torch.nn as nn

# Class - blueprint:
class MyMLP(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super().__init__()
        self.layer1 = nn.Linear(input_dim, hidden_dim)
        self.relu = nn.ReLU()
        self.layer2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = self.layer1(x)
        x = self.relu(x)
        x = self.layer2(x)
        return x

# Object - actual model:
model_a = MyMLP(input_dim=784, hidden_dim=128, output_dim=10)
model_b = MyMLP(input_dim=512, hidden_dim=256, output_dim=10)
# 3 farkli model, ayni class blueprint'inden
```

Mosh'un "farklı öğrenciler yarat" mantığı = ML'de **farklı model konfigürasyonları**.

12.3 `class`, `__init__`, `self`

12.3.1 Class Tanımı

> *"way that I can do that is just by typing sınıf."* — Mosh (Türkçe dublaj), 3:46:43

```
class Student:
    pass
```

İsimlendirme: Class adları `PascalCase`. `Student`, `NeuralNetwork`, `BertTokenizer`.

12.3.2 `__init__` Constructor

> *"I want to create something called an initialize işlevi."* — Mosh (Türkçe dublaj), 3:47:40

```
class Student:
    def __init__(self, name, major, gpa, is_on_probation):
        self.name = name
        self.major = major
        self.gpa = gpa
        self.is_on_probation = is_on_probation
```

`__init__` (init — initialize) = Python'un **constructor**'ı. Otomatik çağrılır.

İki alt çizgi (`__`) — "dunder" (double underscore) — Python'a özel anlam.

12.3.3 `self` — Instance Referansı

`self` her method'un **ilk parametresidir**. Object'in **kendi kendine referansı**.

```
def __init__(self, name, major, gpa, is_on_probation):
    self.name = name          # bu instance'in adi
    ...
```

C++/Java'da `this`; Python'da `self`.

12.3.4 Object Yaratma

```python
# student.py
class Student:
    def __init__(self, name, major, gpa, is_on_probation):
        self.name = name
        self.major = major
        self.gpa = gpa
        self.is_on_probation = is_on_probation

# app.py
from student import Student

student1 = Student("Jim", "Business", 3.1, False)
```

Akış:

1. `Student("Jim", "Business", 3.1, False)` çağrısı.
2. Python yeni `Student` instance yaratır.
3. `__init__(self, "Jim", "Business", 3.1, False)` çağrılır.
4. `__init__` içinde `self.name = "Jim"` …
5. Tamamlanan instance `student1`'e atanır.

`self` parametresi **çağrıda yazılmaz** — Python otomatik geçirir.

12.3.5 Attribute Erişim

```python
student1 = Student("Jim", "Business", 3.1, False)

print(student1.name)              # Jim
print(student1.major)             # Business
print(student1.gpa)               # 3.1
print(student1.is_on_probation)   # False
```

12.3.6 PyTorch'ta `__init__`

! Builder Notu — Modelin İskeleti

```
import torch.nn as nn

class SimpleClassifier(nn.Module):
    def __init__(self, input_dim, num_classes):
        super().__init__()             # nn.Module __init__'ini cagir
        self.feature_extractor = nn.Sequential(
            nn.Linear(input_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
        )
        self.classifier = nn.Linear(128, num_classes)

    def forward(self, x):
        features = self.feature_extractor(x)
        return self.classifier(features)

model = SimpleClassifier(input_dim=784, num_classes=10)
print(model.classifier)     # Linear layer
```

Mosh'un `self.name = name` öğretişi, PyTorch'ta `self.layer = nn.Linear(...)` ile **birebir** aynı yapı. Sadece `name` yerine katman, katmanlar autograd ile gradient hesabına dahil. `super().__init__()` modern Python OOP'sinin standart deyimi. PyTorch'ta **zorunlu**.

12.4 Methods — Class İçinde Fonksiyonlar

Object **veri** taşımakla kalmaz, **iş** de yapar.

12.4.1 Method Tanımı

```
class Student:
    def __init__(self, name, major, gpa, is_on_probation):
        self.name = name
        self.major = major
        self.gpa = gpa
        self.is_on_probation = is_on_probation

    def on_honor_roll(self):
        """GPA 3.5+ ise honor roll'da."""
        return self.gpa >= 3.5

    def graduate(self):
```

```
        """Mezuniyet acıklaması."""
        if self.gpa < 2.0:
            return f"{self.name} mezun olamaz - GPA dusuk"
        elif self.is_on_probation:
            return f"{self.name} probation'da"
        else:
            return f"Tebrikler {self.name}!"
```

Method = class içinde tanımlı fonksiyon. İlk parametre `self`.

12.4.2 Method Çağrısı

```
student1 = Student("Jim", "Business", 3.6, False)

print(student1.on_honor_roll())   # True
print(student1.graduate())        # Tebrikler Jim!
```

`student1.on_honor_roll()` aslında `Student.on_honor_roll(student1)` — Python `student1`'i `self` olarak geçirir.

12.4.3 Method'lar Argüman Alabilir

```
class BankAccount:
    def __init__(self, owner, balance=0):
        self.owner = owner
        self.balance = balance

    def deposit(self, amount):
        if amount <= 0:
            raise ValueError("Amount must be positive")
        self.balance += amount
        return self.balance

    def withdraw(self, amount):
        if amount > self.balance:
            raise ValueError("Insufficient funds")
        self.balance -= amount
        return self.balance

    def transfer(self, other, amount):
        self.withdraw(amount)
        other.deposit(amount)
```

```
deniz = BankAccount("Deniz", 1000)
aylin = BankAccount("Aylin", 500)
deniz.transfer(aylin, 300)
print(deniz.balance)      # 700
print(aylin.balance)      # 800
```

12.4.4 PyTorch'ta `forward()`

Builder Notu — Transformer Forward

```
class MultiHeadAttention(nn.Module):
    def __init__(self, d_model, n_heads):
        super().__init__()
        self.d_model = d_model
        self.n_heads = n_heads
        self.W_q = nn.Linear(d_model, d_model)
        self.W_k = nn.Linear(d_model, d_model)
        self.W_v = nn.Linear(d_model, d_model)
        self.W_o = nn.Linear(d_model, d_model)

    def forward(self, x, mask=None):
        """Tek bir attention block forward."""
        Q = self.W_q(x)
        K = self.W_k(x)
        V = self.W_v(x)

        scores = (Q @ K.transpose(-2, -1)) / (self.d_model ** 0.5)
        if mask is not None:
            scores = scores.masked_fill(mask == 0, float('-inf'))
        attn = scores.softmax(dim=-1)
        return self.W_o(attn @ V)
```

`forward` PyTorch'un **en önemli method'u**. Mosh'un `student.on_honor_roll()` mantığı, transformer'ın `model.forward(x)` mantığıyla **birebir aynı yapıda**: instance state'i kullanarak (`self.W_q`, `self.gpa`) bir hesaplama yap.

12.5 Inheritance — Class'tan Class Türetmek

12.5.1 Motivasyon

```
class Chef:
    def make_chicken(self):
        return "Tavuk pisir"
```

```
    def make_salad(self):
        return "Salata yap"

    def make_special_dish(self):
        return "Bbq tavuk"


class ChineseChef:
    def make_chicken(self):
        return "Tavuk pisir"      # AYNI Chef'le!

    def make_salad(self):
        return "Salata yap"        # AYNI!

    def make_special_dish(self):
        return "Orange chicken"    # FARKLI

    def make_fried_rice(self):
        return "Pirinc kavur"      # YENI
```

Çoğu method **aynı**. Kod tekrarı.

12.5.2 Inheritance ile Çözüm

```
class Chef:
    def make_chicken(self):
        return "Tavuk pisir"

    def make_salad(self):
        return "Salata yap"

    def make_special_dish(self):
        return "Bbq tavuk"


class ChineseChef(Chef):    # Chef'ten miras al!

    # make_chicken, make_salad otomatik geliyor!

    def make_special_dish(self):    # OVERRIDE
        return "Orange chicken"

    def make_fried_rice(self):      # YENI
        return "Pirinc kavur"
```

12.5.3 Override

```
chef = Chef()
chinese = ChineseChef()

print(chef.make_chicken())          # Tavuk pisir
print(chinese.make_chicken())       # Tavuk pisir   (miras)

print(chef.make_special_dish())     # Bbq tavuk
print(chinese.make_special_dish())  # Orange chicken (override)

print(chinese.make_fried_rice())    # Pirinc kavur (yeni)
print(chef.make_fried_rice())       # AttributeError
```

12.5.4 `super()` ile Parent'a Erişim

```
class Animal:
    def __init__(self, name):
        self.name = name
        self.alive = True

    def describe(self):
        return f"{self.name} is alive: {self.alive}"


class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)    # Parent'in __init__'ini cagir
        self.breed = breed

    def describe(self):
        parent_desc = super().describe()
        return f"{parent_desc}, breed: {self.breed}"
```

12.5.5 `isinstance` ile Tip Kontrolü

```
bobby = Dog("Bobby", "Labrador")

isinstance(bobby, Dog)       # True
isinstance(bobby, Animal)    # True (alt-sinif kontrolu)
isinstance(bobby, str)       # False
```

ML kodda:

```
def process_model(model):
    if isinstance(model, nn.Module):
        return model.state_dict()
    elif isinstance(model, dict):
        return model
    else:
        raise TypeError("Unsupported")
```

12.5.6 PyTorch Inheritance

Builder Notu — `class MyBertClassifier(BertModel):`

Bütün PyTorch modelleri `nn.Module`'den inherit eder:

```
import torch.nn as nn

class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = nn.Linear(10, 1)

    def forward(self, x):
        return self.fc(x)

# HuggingFace:
from transformers import BertModel

class MyBertClassifier(BertModel):    # BertModel'i inherit
    def __init__(self, config, num_classes):
        super().__init__(config)
        self.classifier = nn.Linear(config.hidden_size, num_classes)

    def forward(self, input_ids, attention_mask=None):
        outputs = super().forward(input_ids, attention_mask=attention_mask)
        return self.classifier(outputs.pooler_output)
```

`class MyBertClassifier(BertModel):` — HuggingFace'in **milyarlarca parametreli** BERT modelinin tüm özelliklerini al, üzerine kendi classifier'ını ekle.
Mosh'un `class ChineseChef(Chef):` örneği, modern LLM kodlarında **tam olarak bu yapıda**. Sadece `Chef` yerine `BertModel`, `make_special_dish` yerine `forward`.

12.6 Special Methods (Dunders)

Mosh göstermez ama modern Python'un en güçlü özelliklerinden.

12.6.1 `__str__` ve `__repr__`

```
class Student:
    def __init__(self, name, gpa):
        self.name = name
        self.gpa = gpa

    def __str__(self):
        return f"Student({self.name}, GPA={self.gpa})"

s = Student("Jim", 3.5)
print(s)        # Student(Jim, GPA=3.5)
```

12.6.2 `__len__` ve `__getitem__`

```
class Playlist:
    def __init__(self, songs):
        self.songs = songs

    def __len__(self):
        return len(self.songs)

    def __getitem__(self, idx):
        return self.songs[idx]


p = Playlist(["Song 1", "Song 2", "Song 3"])
len(p)        # 3       (__len__ tetiklendi)
p[0]          # Song 1  (__getitem__ tetiklendi)

for song in p:     # iteration otomatik!
    print(song)
```

Python'un duck typing — sıradan obje birden list gibi davranır.

12.6.3 `__call__` — Instance'i Çağrılabilir Yap

```
class Counter:
    def __init__(self):
        self.count = 0

    def __call__(self, increment=1):
```

```
        self.count += increment
        return self.count


c = Counter()
print(c())     # 1   (c() __call__'u tetikler!)
print(c())     # 2
print(c(5))    # 7
```

PyTorch'un kalbi:

```
model = MyModel()
output = model(x)    # __call__ otomatik forward'i cagirir
```

12.6.4 Diğer Yararlı Dunders

Dunder	Tetikleyici	Kullanım
`__init__`	`Cls(...)`	Constructor
`__str__`	`str(obj)`, `print`	Human-readable
`__repr__`	`repr(obj)`, REPL	Debug
`__len__`	`len(obj)`	Uzunluk
`__getitem__`	`obj[i]`	Indexing
`__contains__`	`x in obj`	Membership
`__iter__`	`for x in obj:`	Iteration
`__call__`	`obj(...)`	Çağrı
`__eq__`	`==`	Eşitlik
`__add__`	`+`	Toplama
`__enter__`, `__exit__`	`with obj:`	Context manager

Python'un her operatörü bir dunder'a bağlı.

12.6.5 PyTorch Dataset = Dunders

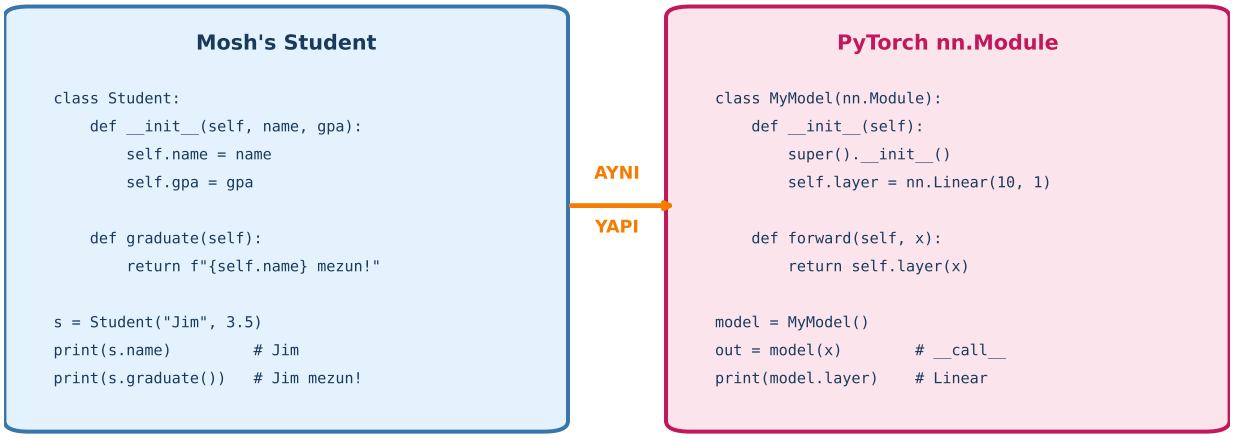
! Builder Notu — DataLoader Magic

```
import torch
from torch.utils.data import Dataset

class MyImageDataset(Dataset):
    def __init__(self, image_paths, labels, transform=None):
        self.image_paths = image_paths
        self.labels = labels
        self.transform = transform

    def __len__(self):                # len(dataset) icin
        return len(self.image_paths)

    def __getitem__(self, idx):       # dataset[i] icin
        path = self.image_paths[idx]
        label = self.labels[idx]

        from PIL import Image
        image = Image.open(path).convert("RGB")

        if self.transform:
            image = self.transform(image)

        return image, label


dataset = MyImageDataset(paths, labels)
print(len(dataset))       # __len__ cagrildi
image, label = dataset[0] # __getitem__ cagrildi

from torch.utils.data import DataLoader
loader = DataLoader(dataset, batch_size=32, shuffle=True)

for batch_images, batch_labels in loader:    # iteration!
    ...
```

PyTorch'un **tüm** DataLoader sistemi `__len__` ve `__getitem__` dunders üzerine kurulu. Mosh göstermez ama bu mekanizma modern ML pipeline'ının temelidir.

12.7 TAM `nn.Module` — Kursun Sentezi

Bu bölüm Ders 12'nin **ve tüm kursun** zirvesi. Mosh'un 8 saatte öğrettiği her şey, modern PyTorch model sınıfında birleşiyor.



Şekil 12.2: Mosh'un Student class'ı → PyTorch nn.Module — sentez

```python
import torch
import torch.nn as nn
import torch.nn.functional as F


class Transformer(nn.Module):
    """Mini transformer encoder. Mosh'un tum derslerinin sentezi."""

    def __init__(self, vocab_size, d_model=512, n_heads=8, n_layers=6, max_len=128):
        # === Ders 12 (OOP): super, __init__ ===
        super().__init__()

        # === Ders 2 (variables, types) ===
        self.d_model = d_model
        self.n_heads = n_heads

        # === Ders 4 (list, tuple) - ModuleList ===
        self.embedding = nn.Embedding(vocab_size, d_model)
        self.pos_embedding = nn.Embedding(max_len, d_model)

        # === Ders 8 (loop), Ders 12 (inheritance) ===
        self.attention_layers = nn.ModuleList([
            MultiHeadAttention(d_model, n_heads) for _ in range(n_layers)
        ])
        self.ff_layers = nn.ModuleList([
            FeedForward(d_model) for _ in range(n_layers)
        ])
```

```
        # === Ders 6 (decision) - condition ile final layer ===
        self.classifier = nn.Linear(d_model, vocab_size)

    def forward(self, x, mask=None):
        """Mosh'un Ders 5 fonksiyonu + Ders 8 dongusu + Ders 12 self."""
        # === Ders 4 (tuple unpacking) ===
        batch, seq_len = x.shape

        # === Ders 2 (arithmetic) ===
        positions = torch.arange(seq_len, device=x.device).unsqueeze(0).expand(batch, -1)

        # === Ders 12 (self.attribute) ===
        x = self.embedding(x) + self.pos_embedding(positions)

        # === Ders 8 (for loop), Ders 12 (zip) ===
        for attn, ff in zip(self.attention_layers, self.ff_layers):
            # === Ders 12 (__call__) - layer(x) ===
            x = x + attn(x, mask)        # residual connection
            x = x + ff(x)

        # === Ders 5 (return) ===
        return self.classifier(x)

    def __repr__(self):
        """Ders 12 (dunder method)."""
        n_params = sum(p.numel() for p in self.parameters())
        return f"Transformer(d_model={self.d_model}, params={n_params:,})"


# === Ders 3 (input, args), Ders 7 (dict config) ===
config = {
    "vocab_size": 50000,
    "d_model": 512,
    "n_heads": 8,
    "n_layers": 6,
}

# === Ders 12 (instantiation) ===
model = Transformer(**config)
print(model)
# Transformer(d_model=512, params=18,832,128)
```

12.7.1 Mosh'un 8 Saatinin Tek Class'ta Sentezi

Bu **tek dosyada**:

Ders	Konsept	Yerleşim
1	print, sıralı yürütme	`print(model)`, kodun akışı
2	değişken, tip	`self.d_model = d_model`
3	input → config dict	`Transformer(**config)`
4	list, tuple unpacking	`ModuleList([...])`, `batch, seq_len = x.shape`
5	def, return	`def forward(...)`, `return self.classifier(x)`
6	if, comparison	forward içinde mask kontrolü
7	dict	`config = {...}`, `Transformer(**config)`
8	for, zip	`for attn, ff in zip(...)`
9	docstring/yorum	`"""..."""`, satır yorumları
10	(try/except, with)	production'da olurdu
11	import	`import torch`, `import torch.nn as nn`
12	class, **init**, self, inheritance, **repr**	**her satır**

Bütün kurs, tek class'ta birleşiyor.

12.7.2 `for batch in dataloader` — Modern AI'nın Tek Cümlesi

```
for batch in dataloader:
    ...
```

Bu **dokuz karakter** modern AI'nın çekirdek mantığını taşır. GPT-4 eğitilirken bu satır milyonlarca kez çağrıldı.

12.8 Karpathy'nin minGPT'si

Builder Notu — Sonraki Adım

Andrej Karpathy'nin GPT'yi sıfırdan yazdığı minGPT projesini incelemek istersen, **her satırı** bu kursta öğrendiğin kavramları kullanır. `class GPT(nn.Module):`, `def __init__`, `def forward`, `for layer in self.blocks:`, `self.attn = MultiHeadAttention(...)`.
Mosh'un kursunu bitirdikten sonra **anlaşılır olacak**.

12.9 Bu Dersin Özeti

1. **Class** = veri tipi blueprint. PascalCase.
2. **`__init__(self, ...)`** = constructor.

3. `self` = instance referansı.
4. `obj = Cls(args)` = object yaratma.
5. `obj.attribute` = erişim.
6. `def method(self, ...)` = method.
7. `class Child(Parent):` = inheritance.
8. `super().__init__(...)` = parent constructor.
9. **Dunders** `__str__`, `__len__`, `__getitem__`, `__call__`.
10. `isinstance(obj, Cls)` = tip kontrolü.
11. `class MyModel(nn.Module):` = modern AI'nın temeli.

! Tek Bir Cümle

Class Python'da kendi veri tipini yaratmanı sağlar — Mosh'un `class Student:` mikro örneği, modern AI'nın `class Transformer(nn.Module):` mimarisinin **birebir** aynı yapıdaki atasıdır; `__init__` constructor, `self.attribute = value` instance state, methods (`def forward(self, x):`), inheritance (`class MyModel(BertModel):`), special methods (`__call__` → `model(x)`, `__getitem__` → `dataset[i]`) — Python OOP'nin tüm kavramları PyTorch'un, transformers'ın, ve modern LLM kodlarının yapı taşlarıdır; Mosh'un sekiz saatlik kursunda öğrettiği **her şey** bir `class Model(nn.Module): def forward(self, x): ...` deyiminde sentez bulur — ve bu noktadan sonra Karpathy'nin minGPT'sini, Hugging Face transformers'ı, PyTorch tutorial'larını okuduğunda **sana hiç yabancı gelmeyecek**.

12.10 Kurs Sonu — Mosh'un Sekiz Saatinin Sentezi

Bu Mosh Hamedani'nin Python for Beginners kursunun **son** dersi.

12.10.1 Kursun 12 Dersinin Final Sentezi

Ders	Konu	ML Köprüsü
1	Kurulum	Python = ML lingua franca
2	Değişkenler	str/int/float = tensor dtype
3	Girdi	LLM prompt template'in atası
4	Listeler ve Tuple	ndarray = Tensor
5	Fonksiyonlar	`nn.Module.forward()`
6	Karar Yapıları	`torch.where`, attention masking
7	Sözlükler/Setler	state_dict, NLP vocabulary
8	Döngüler	`for batch in dataloader`
9	Çevirmen + Yorum	NLP preprocessing, ML docs
10	Hata + Dosyalar	NaN guard, checkpoint persistence
11	Modüller + Pip	ML kütüphane evrenine kapı
12	OOP	`class MyModel(nn.Module):`

12.10.2 Şimdi Ne Yap?

1. Pratik:

- Bu kursun egzersizlerini bitir.
- 5-10 küçük Python projesi yaz.
- Kod kıracak — bu **iyidir**.

2. ML evrenine giriş:

- `pip install numpy pandas matplotlib scikit-learn jupyter`.
- Jupyter notebook'ta gez.
- Kaggle'da bir başlangıç kompetisyonu.

3. Derin öğrenme:

- `pip install torch torchvision`.
- PyTorch'un 60-minute blitz.
- Karpathy'nin Zero to Hero.

4. LLM evreni:

- `pip install transformers`.
- Hugging Face Course.
- Karpathy'nin minGPT projesi.

5. Kardeş kurslar:

Phase 1 serisi:

- MIT 18.06 — Linear Algebra
- Harvard Stat 110 — Probability
- 3Blue1Brown — Calculus
- MIT 6.S191 — Deep Learning

12.11 Egzersizler

Egzersiz 1. Student class + method:

```
class Student:
    def __init__(self, name, major, gpa):
        ???

    def on_honor_roll(self):
        return ???

    def status(self):
        if self.gpa >= 3.5:
            return f"{self.name} (honor roll)"
```

```
        elif self.gpa >= 2.0:
            return f"{self.name} (regular)"
        else:
            return f"{self.name} (probation)"
```

Egzersiz 2. BankAccount + try/except:

```
class BankAccount:
    def __init__(self, owner, balance=0):
        ???

    def deposit(self, amount):
        """amount <= 0 ise ValueError."""
        ???

    def withdraw(self, amount):
        """Yetersiz bakiye -> ValueError."""
        ???

    def transfer(self, other, amount):
        ???
```

Egzersiz 3. Inheritance — Animal / Dog / Cat:

```
class Animal:
    def __init__(self, name, age):
        ???

class Dog(Animal):
    def __init__(self, name, age, breed):
        super().???
        self.breed = breed

    def speak(self):
        return f"{self.name}: Woof!"

class Cat(Animal):
    def __init__(self, name, age, indoor=True):
        super().???
        self.indoor = indoor

    def speak(self):
        return f"{self.name}: Meow!"
```

Egzersiz 4. Vector2D + dunders (`+`, `-`, `*`, `==`):

```python
import math

class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return ???

    def __mul__(self, scalar):
        return ???

    def __eq__(self, other):
        return ???

    def __str__(self):
        return f"Vector2D({self.x}, {self.y})"

    def magnitude(self):
        return math.sqrt(self.x**2 + self.y**2)
```

NumPy/PyTorch tensor matematik operatörlerinin ilkel versiyonu.

Egzersiz 5. (FİNAL — mini Model hiyerarşisi)

```python
class Model:
    def __init__(self, name):
        self.name = name
        self.trained = False

    def train(self, data):
        raise NotImplementedError

    def predict(self, x):
        raise NotImplementedError

    def __repr__(self):
        return f"{self.__class__.__name__}(name={self.name!r}, trained={self.trained})"


class LinearModel(Model):
    def __init__(self, name="Linear"):
        super().???
        self.m = None
        self.b = None
```

```
    def train(self, data):
        # Basit lineer regresyon
        xs = [x for x, _ in data]
        ys = [y for _, y in data]
        n = len(data)
        x_mean = sum(xs) / n
        y_mean = sum(ys) / n
        # Egim:
        num = sum((x - x_mean) * (y - y_mean) for x, y in data)
        den = sum((x - x_mean) ** 2 for x in xs)
        self.m = num / den
        self.b = y_mean - self.m * x_mean
        self.trained = True

    def predict(self, x):
        if not self.trained:
            raise RuntimeError("Henuz egitilmedi")
        return self.m * x + self.b


# Test:
data = [(1, 2), (2, 4), (3, 6), (4, 8)]   # y = 2x

lin = LinearModel()
lin.train(data)
print(lin)                # LinearModel(name='Linear', trained=True)
print(lin.predict(5))     # ~10 (y = 2*5)
```

Bu egzersiz **scikit-learn** API'sinin mini hâlidir. Mosh'un öğrettiği OOP, gerçek ML kütüphanelerinin çekirdek API tasarımıdır.

> **!** Bu kursun final cümlesi
>
> Mosh'un Ders 1'de "Python öğrenmek = ML ekosisteminin bilet kasasına girmek" dediğimiz şey **artık gerçek**. Ders 2'de değişken, Ders 3'te input, Ders 4'te list/tuple, Ders 5'te fonksiyon, Ders 6'da if/else, Ders 7'de dict, Ders 8'de for-batch-in-dataloader, Ders 9'da preprocessing, Ders 10'da defansif kod, Ders 11'de pip install, ve Ders 12'de class — her satır seninle bugünden itibaren modern AI mühendisliği yolunda. `class MyModel(nn.Module): def __init__(self): super().__init__(); self.layer = nn.Linear(...); def forward(self, x): return self.layer(x)` artık sana yabancı değil — Mosh'un öğrettiği `class Student: def __init__(self, name): self.name = name; def graduate(self): return f"{self.name} mezun oldu"` ile **birebir** aynı yapı, sadece tensor ve gradient katılmış. Sonraki adım: `pip install torch transformers`, ve Karpathy'nin minGPT'sine git. Mosh'un sekiz saat boyunca öğrettiği her temel kavram, modern AI'nın yapı taşıdır — ve sen artık bunları biliyorsun.