1.Ünite - Python ile Programlamaya Giriş

Python'un Genel Özellikleri

- **Python**, Guido van Rossum tarafından geliştirildi ve ilk sürümü **1990** yılında yayımlandı.
- İsmini, geliştiricinin hayranı olduğu **"Monty Python's Flying Circus"** adlı komedi dizisinden aldı.
- **Statista (2025)** verilerine göre 2025 yılında en popüler üçüncü programlama dili oldu (%51).

- **Nesneye Yönelimli Programlama (OOP)** dilidir; her şey bir **nesnedir (object)**.
- Nesneler; **özellikler (attributes)** ve **metotlardan (methods)** oluşur, nokta (.) sözdizimiyle erişilir.
- OOP'nin temel ilkeleri:
  - **Miras alma (inheritance)**
  - **Çok biçimlilik (polymorphism)**
  - **Kapsülleme (encapsulation)**
- Amaç, veriyi ve fonksiyonları tek bir birimde birleştirerek dış erişimi sınırlandırmaktır.
- **C, C++, C#** gibi diller **derleyici (compiler)** kullanır; kodu bir defada çevirir.
- **Python** ise **yorumlayıcı (interpreter)** kullanır; kodu **satır satır çalıştırır**, hata ayıklama kolaydır.
- **Derleyiciler:**
  - Kodun tamamını analiz eder, hatayı sonradan bildirir.
  - Analiz yavaş ama çalıştırma hızlıdır.
- **Yorumlayıcılar:**
  - Hataları anında yakalar, çalışmayı durdurur.
  - Hata tespiti kolaydır.

Python'un Temel Özellikleri

- Farklı veri tipleri desteklenir: **sayılar, metin (string), listeler, sözlükler (dict)**.
- **Nesneye yönelimli programlama** desteklenir (sınıflar, çoklu kalıtım).
- Kod **modüller** ve **paketler** halinde organize edilir.
- **İstisna yönetimi (exception handling)** ile daha temiz hata işleme sağlanır.
- **Güçlü ve dinamik tip sistemi** bulunur (uyumsuz tipler istisna üretir).
- **Generators** ve **list comprehensions** gibi gelişmiş yapılar içerir.
- **Otomatik bellek yönetimi** vardır (manuel bellek ayırma gerekmez).

Python'un Lisans ve Topluluk Özellikleri

- **Özgür ve açık kaynaklı (open-source)** bir dildir.
- **Ücretsizdir**, indirilebilir, değiştirilebilir ve dağıtılabilir.
- Kodlar **Kaggle**, **GitHub** gibi platformlarda paylaşılabilir.
- Geniş bir topluluk tarafından desteklenir, sürekli gelişmektedir.

Python'ın Kullanım Alanları

- **Yapay Zekâ (AI) ve Makine Öğrenmesi (ML):**
  TensorFlow, PyTorch, Scikit-learn gibi kütüphanelerle model geliştirme, veri analizi.
- **Veri Bilimi ve Analitiği:**
  NumPy, Pandas, Matplotlib, Seaborn gibi araçlarla veri toplama, temizleme ve görselleştirme.
- **Web Geliştirme:**
  Django, Flask, FastAPI gibi framework'lerle dinamik web siteleri ve API'ler geliştirme.
- **Otomasyon ve Betik Yazımı (Scripting):**
  Tekrarlayan görevleri otomatikleştirme (örneğin dosya işlemleri, web scraping).
- **Veritabanı Uygulamaları:**
  SQLite, MySQL, PostgreSQL, MongoDB gibi sistemlerle veri tabanı yönetimi.
- **Siber Güvenlik ve Etik Hacking:**
  Penetrasyon testleri, zafiyet taramaları ve güvenlik otomasyonu için (örneğin: Scapy, Requests).
- **Bilimsel ve Matematiksel Hesaplamalar:**
  SciPy, SymPy, NumPy gibi kütüphanelerle sayısal analiz ve mühendislik hesaplamaları.
- **Biyoenformatik ve Sağlık:**
  Genetik veri analizi, protein modelleme, medikal görüntü işleme.
- **Oyun Geliştirme:**
  Pygame, Panda3D gibi kütüphanelerle 2D/3D oyunlar geliştirme.
- **Masaüstü Uygulama Geliştirme:**
  Tkinter, PyQt, Kivy gibi araçlarla GUI tabanlı uygulamalar geliştirme.
- **Eğitim ve Programlama Öğretimi:**
  Kolay sözdizimi sayesinde yeni başlayanlar için ideal bir öğretim dili.
- **Mobil Uygulama Geliştirme:**
  Kivy, BeeWare gibi kütüphanelerle platformlar arası mobil uygulamalar geliştirme.
- **Bulut Bilişim ve DevOps:**
  AWS, Azure SDK'ları, otomasyon script'leri, altyapı yönetimi.
- **IoT (Nesnelerin İnterneti):**
  Raspberry Pi, MicroPython ile sensör verisi toplama ve cihaz kontrolü.

Python Kurulumu

İşte **Python kurulumu** işletim sistemlerine göre sade ve net biçimde

---

Windows

1. python.org/downloads adresine git.
2. "**Download Python 3.x.x**" butonuna tıkla.
3. İndirdiğin .exe dosyasını çalıştır.
4. Kurulum ekranında mutlaka **"Add Python to PATH"** kutusunu işaretle.
5. "Install Now" seçeneğine tıkla ve kurulumun tamamlanmasını bekle.
6. Kurulumdan sonra Başlat menüsünden **CMD (Komut İstemi)** aç →

   ```
   python --version
   ```

   yaz ve kurulumun başarılı olup olmadığını kontrol et.

---

macOS

1. macOS genellikle eski bir Python sürümüyle gelir; güncel sürüm için:

python.org/downloads adresine git.

2. .pkg dosyasını indir ve kurulum sihirbazını takip et.
3. Terminal'i aç ve sürümü kontrol et:

   python3 --version

4. İstersen **Homebrew** ile de kurabilirsin:

   brew install python

Linux (Ubuntu / Debian / Arch / Fedora vs.)

Python çoğu Linux dağıtımında önceden kurulu gelir.
Yine de güncellemek veya manuel kurmak için:

Ubuntu / Debian:

sudo apt update
sudo apt install python3

Fedora:

sudo dnf install python3

Arch Linux:

sudo pacman -S python

Kurulumu doğrulamak için:

python3 --version

---

pip (Python Paket Yöneticisi) versiyonunu kontrol etmek için

pip --version

Python ile "Hello World"

1. Terminal (veya Komut İstemi) aç.
2. python (veya bazı sistemlerde python3) yaz ve Enter'a bas.
3. Açılan Python etkileşimli kabuğuna şunu yaz:

   python print("Hello, World!")

4. Enter'a bastığında konsolda şu sonucu göreceksin:

   Hello, World!

Alternatif:
Bir .py dosyası oluşturup da yazabilirsin.

1. Yeni bir dosya oluştur: hello.py
2. Dosyanın içine yaz:

   python print("Hello, World!")

3. Terminalde çalıştır:

   python hello.py

4. Çıktı:

   Hello, World!

Python Geliştirme Ortamları.

IDE'ler

- **PyCharm**
- **Spyder**
- **Thonny**
- **Python IDLE**
- **Jupyter Notebook** / **JupyterLab**
- **Anaconda Navigator**

Editörler

- **Visual Studio Code (VS Code)**
- **Sublime Text**
- **Atom** → Artık geliştirilmiyor.
- **Notepad++**
- **Vim** / **Neovim**

Python da Modül, Paket, Kütüphane ve Çerçeve

Modül (Module)

- .py uzantılı dosyalara kaydedilmiş, birbiriyle ilişkili bir grup Python kodu.
- Fonksiyonlar, sınıflar veya değişkenler yerleştirilebilir.
- **Örnekler:** random, math, html, datetime, re.
- **Çağırma Yöntemleri:**
  - **Tümünü Çağırma:** python import math print(math.pi)
  - **Belirli Bir Unsuru Çağırma:** python from math import pi print(pi)
  - **Yeniden Adlandırarak Çağırma:** python import math as m print(m.pi)
- **Kullanım Örneği (OS Modülü):** Çalışma dizini işlemlerini yönetmek için os modülü kullanılır (os.getcwd(), os.chdir()). python import os calismaKlasoru = os.getcwd() # Mevcut dizini al print(calismaKlasoru) os.chdir("D://auzef") # Yeni dizini ayarla (dosya yolunda `//` kullanılır) print(os.getcwd())

Paket (Package)

- Temel olarak bir **modül koleksiyonunun bir dizinidir**.
- **Yapı:** Mantıksal olarak ilişkili modülleri bir klasör hiyerarşisi altında gruplar (tıpkı sabit sürücüdeki klasörler/alt klasörler gibi).
- **Amaç:** Modül ad alanının (namespace) hiyerarşik yapısına olanak sağlamak.
- **Örnekler:** xml, email, numpy.linalg, pandas.plotting.
- **Çağırma Yöntemi:** python import paket_adı from paket.altmodül import fonksiyon

---

Kütüphane (Library)

- Yeniden kullanılabilir bir kod topluluğuna atıfta bulunan geniş anlamlı bir terimdir.
- **Yapı:** Genellikle ilgili **modüller ve paketlerden** oluşan bir koleksiyonu içerir.
- **Kullanım:** Python'da "paket" ve "kütüphane" terimleri sıkça birbirinin yerine kullanılabilir, ancak genellikle**kütüphane, bir paketler koleksiyonu** olarak görülür.
- **Örnekler:** NumPy, pandas, Matplotlib, PyTorch, pygame.
- **Çağırma Yöntemi:** python import kütüphane_adı

Çerçeve (Framework)

- Programcıların geliştirme sürecini hızlandırmaya yardımcı olan bir**modüller ve paketler koleksiyonudur** (kütüphanelere benzer).
- Kütüphaneler belirli işlemleri gerçekleştiren paketleri içerirken, **çerçeveler uygulamanın temel akışını ve mimarisini**

içerir. Uygulamanın genel yapısını (iskeleyi) belirler.
- **Örnekler:** django, flask, bottle.

2.Ünite - Python'da Değişkenler ve Veri Tipleri

Değişkenler

- **Değişken (variable):** Bellekte veri saklamak için kullanılan isimdir.
  Örn: int a = 25 → a isimli bir bellek alanına 25 değeri atanmıştır.
- **Atama (assignment):** = işareti matematikteki eşitlik değil, **değer atamayı** temsil eder.
  - Çoğu dilde atama **sağdan sola** yapılır (Python dahil).
  - R dilinde farklı yazımlar da mümkündür (a = 25, 25 -> a, a <- 25).
- **Python'da:**
  - Değişken tanımlanırken veri tipi **önceden belirtilmez**, **dinamik olarak** belirlenir.
  - a aslında bir **işaretçidir (pointer)** → bellekteki bir veriyi işaret eder.
  - Atanabilen değerlere **literal** denir:

    python a = 25 # int a = "25" # str a = 2.5 # float
- **Noktalı virgül (;) kullanımı:**
  - C, C++, Java gibi dillerde satır sonunu belirtir.
  - Python'da zorunlu değildir, sadece **aynı satırda birden fazla ifade** yazmak için kullanılır.

    python a = 10; b = 9; c = 8
- **Yorum satırları (comments):**
  - Tek satırlık yorum: # ile başlar.
  - Çok satırlı yorum: Üç çift tırnak (""" ... """) arasına yazılır.

    ```python

    # Bu bir yorum satırıdır

    """ @title: Değişkenler ve Veri Tipleri @author: ... """ ```

**Özetle**

- Python'da değişkenler önceden tanımsızdır, tipi dinamik olarak belirlenir; noktalı virgül isteğe bağlıdır; yorum satırları ise kodun okunabilirliğini artırmak için kullanılır.

### Değişken Tanımlama Kuralları

- Harf veya alt çizgi (_) ile başlamalıdır.
- **Sayı ile başlayamaz.**
- Sadece **alfanümerik karakterler** ve _ kullanılabilir (A–Z, a–z, 0–9, _).
- **Türkçe karakter** (ç, ğ, ı, ö, ş, ü) kullanılmamalıdır.
- **Boşluk** içeremez (ilk sayi hatalıdır).
- **Kısa çizgi (-)** kullanılamaz, çünkü işlem operatörüdür (ilk-sayi hatalıdır).
- **Python anahtar kelimeleri** değişken adı olamaz (if, elif, for, while, sum vb.).

---
    ```

✕ Hatalı değişken örnekleri

```
086_sayi = 1    # sayı ile başlıyor
ilk-sayi = 81   # kısa çizgi içeriyor
ilk sayi = 1903 # boşluk içeriyor
elif = 37       # anahtar kelime kullanılmış
```

Doğru değişken örnekleri

```
sayi = 1903
_sayi = 1903
ilkSayim = 1903
ilk_sayi = 1903
ilkSayim1 = 1903
```

- **Python büyük/küçük harfe duyarlıdır (case sensitive):**
  ilkSayi ≠ ilksayi
- **Çoklu atama** yapılabilir:

  python a, b, c, d = 5, 10, 20, 25

  Bu, birden fazla değişkene tek satırda değer atamayı sağlar.

Özetle

- Python'da değişken isimleri açık, kurallara uygun ve anlamlı olmalıdır. Harf veya_ ile başlamalı, anahtar kelimelerden kaçınılmalı ve istenirse tek satırda birden fazla değişken tanımlanabilir.

Bellek Yönetimi(Memory Management)

- Bilgisayarın belleği, bir kitabın sayfalarına benzetilebilir. Veriler bellekte saklanır, gerekirse silinir ve yeni verilere yer açılır. Bu işlemleri **işletim sistemi** yürütür, ancak Python'da bu yönetimi **Python yorumlayıcısı (CPython)** üstlenir.
- **Bellek tahsisi (Memory Allocation):**
  Bellekte veri saklamak için bir alan ayrılması işlemidir.
  - C dilinde:

    c int x = 10;

    → "x" için bir bellek alanı ayrılır.
  - Python'da her şey bir **nesnedir (object)**.

- **Python'da her şey nesnedir:**
  Her değişken bir **nesne** olarak kabul edilir.

  python x = 10 ad = "Elif" isinstance(x, object) # True isinstance(ad, object) # True

  Her Python nesnesi üç bilgi içerir:

  1. **type (veri tipi)**
  2. **value (değer)**
  3. **reference count (referans sayısı)**

- **Referans sayısı (Reference Count):**
  Bir nesnenin sistemde kaç kez kullanıldığını gösterir.
  - Yeni bir değişken aynı nesneye atıfta bulunursa referans sayısı artar.

  python x = 10 y = x id(x) == id(y) # True → aynı bellek adresi
  - x'in değeri değişirse, yeni bir nesne oluşur ve artık farklı adresi gösterir.

  python x = x + 10 id(x) != id(y)

- **Intern Nesneler (Intern Objects):**
  Python, sık kullanılan bazı nesneleri bellekte **önceden saklar**.
    - CPython 3.7'de:
        - **-5 ile 256** arasındaki sayılar
        - **Sadece ASCII karakterleri içeren metinler**
          otomatik olarak bellekte tutulur.

  Bu nedenle aynı değere sahip değişkenler aynı bellek adresini paylaşır:

  python a = 200; b = 200 id(a) == id(b) # True c = 257; d = 257 id(c) == id(d) # False

  Aynı durum bazı kısa metinler için de geçerlidir:

  python metin1 = "Merhaba" metin2 = "Merhaba" id(metin1) == id(metin2) # True metin3 = "Merhaba!" metin4 = "Merhaba!" id(metin3) == id(metin4) # False

---

Özetle

- Python, belleği otomatik yönetir; her şey nesne temellidir. Aynı değere sahip küçük sayılar ve basit metinler bellekte **tek kopya** olarak tutulur (interning). Bu sistem, hem bellek tasarrufu hem de performans artışı sağlar.

Python Dilinde Temel Veri Tipleri

1. Sayılar (Numbers)

- **int**: Tam sayılar

python x = 10 y = -3 * **float**: Ondalıklı sayılar

python pi = 3.14 sıcaklık = -12.5 * **complex**: Karmaşık sayılar

python z = 2 + 3j

2. Metinler (Strings) Karakter dizileridir.

```
ad = "Ahmet"
mesaj = 'Merhaba Dünya'
print(ad[0])  # 'A'
```

3. Boolean (Mantıksal) True veya False değerleri alır.

```
a = True
b = False
print(5 > 3)  # True
```

4. Listeler (Lists) Birden fazla değeri sıralı tutar, değiştirilebilir.

```
meyveler = ["elma", "muz", "kiraz"]
meyveler.append("armut")
```

5. Demetler (Tuples) Listelere benzer ama değiştirilemez.

```
renkler = ("kırmızı", "mavi", "yeşil")
```

6. Kümeler (Sets) Sırasız, benzersiz öğeler içerir.

```
sayilar = {1, 2, 3, 3}
print(sayilar)  # {1, 2, 3}
```

7. Sözlükler (Dictionaries) Anahtar-değer (key-value) çiftlerinden oluşur.

```
öğrenci = {"ad": "Ahmet", "yaş": 21}
print(öğrenci["ad"])
```

Python dinamik tipli olduğu için değişkenin tipi sonradan değiştirilebilir:

```
x = 5
x = "Merhaba"
```

Python Temel Veri Tipleri Özeti Tablosu

Veri Tipi	Açıklama	Örnek	Çıktı / Not
int	Tam sayılar	x = 10	type(x) → <class 'int'>
float	Ondalıklı sayılar	pi = 3.14	type(pi) → <class 'float'>
complex	Karmaşık sayılar	z = 2 + 3j	z.real → 2, z.imag → 3
str	Metin (karakter dizisi)	ad = "Ahmet"	ad[0] → 'A'
bool	Mantıksal değer	a = True	5 > 3 → True
list	Sıralı, değiştirilebilir koleksiyon	liste = [1, 2, 3]	liste.append(4) → [1, 2, 3, 4]
tuple	Sıralı ama değiştirilemez koleksiyon	t = (1, 2, 3)	t[1] → 2
set	Sırasız, benzersiz öğeler	s = {1, 2, 2, 3}	{1, 2, 3}
dict	Anahtar–değer yapısı	d = {"ad": "Ahmet", "yaş": 21}	d["ad"] → "Ahmet"

Python Kaçış (Escape) Karakterleri

- Kaçış karakterleri, metin içinde özel anlamlı karakterleri göstermek veya biçimlendirme yapmak için kullanılır.

Kaçış Karakteri	Anlamı	Örnek Kod	Çıktı
\n	Yeni satır	print("Merhaba\nDünya")	Merhaba Dünya
\t	Sekme (tab boşluğu)	print("Ad:\tAhmet")	Ad: Ahmet
\\	Ters eğik çizgi (\)	print("C:\\Users\\Ahmet")	C:\Users\Ahmet
\'	Tek tırnak (')	print('Ahmet\'in kalemi')	Ahmet'in kalemi
\"	Çift tırnak (")	print("Ahmet\"in evi")	Ahmet"in evi
\r	Satır başına dön (carriage return)	print("Merhaba\rDünya")	Dünya
\b	Geri silme (backspace)	print("Ahmet\b!")	Ahme!
\f	Sayfa sonu (form feed)	print("A\fB")	A↡B *(bazı ortamlarda görünmez)*
\a	Uyarı sesi (beep)	print("\a")	🔊 *(bazı sistemlerde ses çıkarır)*
\uXXXX	Unicode karakter (16 bit)	print("\u03A9")	Ω
\UXXXXXXXX	Unicode karakter (32 bit)	print("\U0001F600")	😀

Python'da Operatörler ile İşlemler

1. Matematiksel Operatörler

Operatör	Açıklama	Örnek	Çıktı
+	Toplama	5 + 3	8
-	Çıkarma	5 - 3	2
*	Çarpma	5 * 3	15
/	Bölme	5 / 2	2.5
//	Tam bölme	5 // 2	2
%	Modül (kalan)	5 % 2	1
**	Üs alma	2 ** 3	8

2. Atama Operatörleri

Operatör	Açıklama	Örnek
=	Atama	x = 5
+=	Topla ve ata	x += 3 → x = x + 3
-=	Çıkar ve ata	x -= 2 → x = x - 2
*=	Çarp ve ata	x *= 4 → x = x * 4
/=	Böl ve ata	x /= 2 → x = x / 2
//=	Tam böl ve ata	x //= 3
%=	Mod al ve ata	x %= 2
**=	Üs al ve ata	x **= 3

3. Karşılaştırma Operatörleri

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

4. Mantıksal Operatörler

Operatör	Açıklama	Örnek	Çıktı
and	Ve	True and False	False
or	Veya	True or False	True
not	Değil	not True	False

5. Üyelik ve Kimlik Operatörleri

Operatör	Açıklama	Örnek	Çıktı
in	Eleman içeriyor mu?	'a' in 'abc'	True
not in	Eleman içermiyor mu?	'd' not in 'abc'	True
is	Aynı nesne mi?	a is b	True/False
is not	Farklı nesne mi?	a is not b	True/False

Açık ve Örtülü Veri Tipi Dönüşümü

Python'da **tür değişimi (type conversion)** ikiye ayrılır; - **örtülü (implicit)** - **açık (explicit)**

1. Örtülü Tür Değişimi (Implicit Type Conversion)
Python bazı türleri **kendisi otomatik olarak dönüştürür**. Genellikle sayısal türler arasında olur.

```
x = 5      # int
y = 2.5    # float
z = x + y  # Python otomatik olarak int'i float'a çevirir
print(z)   # 7.5
print(type(z))  # <class 'float'>
```

Yani türler uyumluysa, Python hata vermeden uygun olan tipe çevirir.

2. Açık Tür Değişimi (Explicit Type Conversion)
Programcı türü **kendisi dönüştürür** — int(), float(), str() gibi fonksiyonlarla.

```
# string → int
x = "10"
y = int(x)
print(y + 5)  # 15

# float → int
z = int(3.9)
print(z)  # 3

# int → string
a = str(123)
print(a + "4")  # "1234"
```

Özetle:

- **Örtülü dönüşüm:** Python kendisi yapar.
- **Açık dönüşüm:** Programcı yapar.

Tür	Açıklama	Örnek Kod	Çıktı	Açıklama

Tür	Açıklama	Örnek Kod	Çıktı	Açıklama

Örtülü (Implicit)	Python'un otomatik yaptığı dönüşüm	`x = 5`
		`y = 2.5`
		`z = x + y`
			`z = 7.5`	`int` → `float` otomatik dönüşür

Açık (Explicit)	Programcının elle yaptığı dönüşüm	`x = "10"`
		`y = int(x)`
		`print(y + 5)`
			`15`	`str` → `int` dönüşümü

Açık (Explicit)	`float()` ile dönüştürme	`a = float(5)`	`5.0`	`int` → `float`
Açık (Explicit)	`str()` ile dönüştürme	`b = str(123)`	`"123"`	`int` → `str`
Açık (Explicit)	`int()` ile kesirli sayıyı tam yapma	`c = int(3.9)`	`3`	Ondalık kısmı atılır

- **Not:** Örtülü dönüşüm sadece **uyumlu türler** arasında olur.
- Örneğin `"5" + 3` hatadır, çünkü string ve int otomatik çevrilemez.

Değişir ve Değişmez Nesneler

- Python'da veri tipleri **değişir (mutable)** ve **değişmez (immutable)** olarak ikiye ayrılır:
- **Değişmez (Immutable):** Oluşturulduktan sonra içeriği değiştirilemez. Yeni değer verilirse aslında **yeni bir nesne** oluşur.
  Örnek: `int`, `float`, `complex`, `bool`, `str`, `tuple`
- **Değişir (Mutable):** Oluşturulduktan sonra içeriği değiştirilebilir.
  Örnek: `list`, `set`, `dict`

Kategori	Veri Tipi	Değiştirilebilir mi?	Örnek	Açıklama
Immutable (Değişmez)	`int`	✗ Hayır	`x = 5` → `x = 6` yeni nesne oluşturur	Sayılar değiştirilemez
Immutable (Değişmez)	`float`	✗ Hayır	`pi = 3.14`	Değeri değişirse yeni nesne olur
Immutable (Değişmez)	`complex`	✗ Hayır	`z = 2 + 3j`	Karmaşık sayılar da değişmezdir
Immutable (Değişmez)	`bool`	✗ Hayır	`a = True`	Boolean değer değiştirilemez
Immutable (Değişmez)	`str`	✗ Hayır	`s = "Ahmet"` → `s[0] = "B"` hata verir	Metinler değiştirilemez
Immutable (Değişmez)	`tuple`	✗ Hayır	`t = (1, 2, 3)`	Eleman eklenemez veya silinemez
Mutable (Değişir)	`list`	✓ Evet	`l = [1,2]; l.append(3)`	Liste içeriği değiştirilebilir
Mutable (Değişir)	`set`	✓ Evet	`s = {1,2}; s.add(3)`	Küme elemanları değiştirilebilir
Mutable (Değişir)	`dict`	✓ Evet	`d = {"a":1}; d["b"]=2`	Anahtar-değer çiftleri eklenebilir

Özet

- **Değişmez tipler:** yeni değer → yeni nesne.
- **Değişir tipler:** aynı nesne → içeriği değiştirilebilir.

3.Ünite - Veri Yapıları

Python'daki temel veri tiplerinden sonra gelen **bileşik veri yapıları**nı tanıtıyor. Bu yapılar, birden fazla veriyi tutabilen "kaplar"dır.

Python'daki başlıca bileşik veri yapıları:

- **List (liste):** Köşeli parantez `[ ]` ile oluşturulur, değiştirilebilir.
- **Tuple (demet):** Yuvarlak parantez `( )` ile oluşturulur, değiştirilemez.
- **Set (küme):** Süslü parantez `{ }` ile oluşturulur, elemanlar tekrarlanmaz.
- **Dictionary (sözlük):** Anahtar-değer çiftleri `{key: value}` şeklindedir.

Lists

```
names = ["Ahmet", "Mehmet", "Ali", "Muhammed"]
print(type(names)) # list
print(len(names)) # 4
```

İndexing

```
numbers = [1, 3, 5, 9, 12, 45, 67]
print(numbers[0]) # 1
print(numbers[1]) # 3
print(numbers[3]) # 9
print(numbers[-1]) # 67 - listenin sonundaki elemanı return eder.
print(numbers[-3]) # 12
print(numbers[2:]) # [5, 9, 12, 45, 67] | 2. indexten başlar listenin sonunda kadar return eder .
print(numbers[-3:]) # [12, 45, 67] | -3. indexten başlar listenin sonunda kadar return eder.
print(numbers[0:5:2]) # [1, 5, 12] | 0-5 indexler arasındaki 2'şer atlayrak return eder.
print(numbers[::-1]) # [67, 45, 12, 5, 3, 1] | listeyi ters çevrip return eder
```

Listeler tek veri tipinden oluşabileceği gibi, birbirinden farklı veri tipinde nesnelere sahip olabilir.

python a = [True, False, False, True] print(a) # [True, False, False, True] b = ["A" True, 1, 1.5, [0, 3, 6, 9]] print(b) # ["A" True, 1, 1.5, [0, 3, 6, 9]]

List ait bazı methodlar

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

numbers.append(11) # listenin sonuna eleman ekler.
print(numbers) # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

numbers.remove(1) # listeden eleman siler.
print(numbers) # [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

numbers.pop() # listenin sonundaki elemanı siler.
print(numbers) # [2, 3, 4, 5, 6, 7, 8, 9, 10]

print(numbers.index(5)) # 3 | girilen sayının listedeki indexini return eder.

numbers.sort() # listeyi sıralamak için kullanılır. Default olarak artan biçimde sıralar.
print(numbers) # [2, 3, 4, 5, 6, 7, 8, 9, 10]

numbers.reverse() # listeyi tersine çevirir.

print(numbers) # [10, 9, 8, 7, 6, 5, 4, 3, 2]

del numbers # listeyi siler.
```

Tuples

- Tuple birçok yönden listelere benzemektedir; ancak köşeli parantezler yerine yuvarlak parantezlerle tanımlanırlar. Ayrıca herhangi bir parantez kullanmadan da tanımlanabilirler.
- Tuples hakkında unutulmaması gereken en önemli özelliklerden biri değişemez (immutable) olduklarıdır.

```
tuple1 = ("a", "b", "c", "d", 0, 1, 2, 3)
tuple2 = "Ahmet", "Mehmet", "Muhammed"

print(tuple1[0]) # a
print(tuple1[3]) # d
print(tuple1[-1]) # 3

print(tuple2[0]) # Ahmet
print(tuple2[2]) # Muhammed
```

- Bir tuple tanımlarken parantezin içindeki elemanları "," (virgül) ile ayrılması gerek.

```
a = ("Ahmet")
print(type(a)) # str

b = ("Ahmet",)
print(type(b)) # tuple
```

Tuple ait bazı methodlar

`

```
tuple1 = ("a", "b", "c", "d", 0, 1, 2, 3)

del tuple1 # tuple siler
len(tuple1) # tuple eleman sayısını return eder
tuple.count(b) # bir elemanın tuple içinde kaç defa tekrarlandığı return eder.
tuple(sorted(tuple1)) # tuple ters çevirir.
```

Dicts

- Dict (key:value) prensibini kullanan son derece esnek eşleme yapılardır.
- Kıvırcık parantezler arasına tanımlanırlar, elemanlar virgül ile ayrılır.
- Dict boyutu ne olursa olsun rastgele öğelere erişim çok hızlıdır.

```
dict1 = {"name": "Ahmet", "age": 25}
print(dict1) # {"name": "Ahmet", "age": 25}
print(dict1[name]) # Ahmet
print(dict1.get(name)) # Ahmet

print(list(dict1.keys())) # ["name", "age"]
print(list(dict1.values())) # ["Ahmet", "25"]

dict1["hobby"] = "Music"
print(dict1) # {"name": "Ahmet", "age": 25, "hobby": "Music"}

del dict1[hobby] # dict1 den hobby siler.

dict1.pop("age") # dict deki son elemanı siler ve dict return eder

dict1.popitem() # dict deki son elemanı return eder.

dict1.items() # dict deki tüm elemanları listeler
```

Sets

kümeler (set), **benzersiz** ve **sırasız** elemanlardan oluşan **değiştirilebilir** veri yapılarıdır.
Küme işlemleri matematikteki gibi yerleşik metotlarla yapılır:

- **Eleman ekleme:** add()
- **Eleman silme:** pop() (rastgele bir elemanı siler)

Örnek kümeler:

```
asalSayilar = {2, 3, 5, 7, 11, 13, 17, 19}
tekSayilar = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19}
```

Temel işlemler:

- **Kesişim:** asalSayilar.intersection(tekSayilar) veya asalSayilar & tekSayilar
- **Birleşim:** asalSayilar.union(tekSayilar) veya asalSayilar | tekSayilar
- **Fark:** asalSayilar.difference(tekSayilar) veya asalSayilar - tekSayilar

Bu işlemler, kümeler arasında ortak, tüm veya farklı elemanları bulmayı sağlar.

4.Ünite - Koşullu İfadeler

Python'da **koşul ifadeleri (if statements)**, programın belirli durumlarda farklı işlemler yapmasını sağlar.

Temel yapı:

```
if koşul:
    işlem
```

Koşul **True** ise blok çalışır, **False** ise atlanır.
Girintiler (indentation) çok önemlidir; if bloğundaki tüm satırlar aynı hizada olmalıdır.

Örnekler:

- **Tek koşul:**

  ```python x = 10 y = 10

  if x == y: print("Eşit") ```

- **Kısa biçim (short-hand):**

  ```python x = 10 y = 10

  if x == y: print("Eşit") ```

- **Birleşik koşullar:** and, or kullanılır.

  ```python x = 10 y = 10

  if x == y and x > 0 and y > 0: print("Eşit ve pozitif") ```

- **if–else:**

  ```python x = 15 y = 10

  if x > y: print("x büyük") else: print("x küçük veya eşit") ```

- **Kısa if–else:**

  ```python x = 10 y = 15

  print("x büyük") if x > y else print("x küçük veya eşit") ```

- **elif (çoklu koşul):**

  ```python x = 10 y = 15

  if x > y: print("büyük") elif x < y: print("küçük") else: print("eşit") ```

- **İç içe if (nested if):**

  ```python x = 12

  if x < 20: if x > 0: print("0 < x < 20") else: print("x <= 0") else: print("x >= 20") ```

**Kısaca:**
if bir koşulu kontrol eder, elif başka durumları test eder, else tüm kalan durumları yakalar.
Koşullar mantıksal değer döndürür ve program akışını buna göre yönlendirir.

match-case, Python 3.10 ile gelen **yeni bir kontrol yapısıdır** ve diğer dillerdeki *switch-case* mantığına benzer.
Ama aslında bundan daha güçlüdür; sadece sayıları ya da metinleri değil, **desenleri (patterns)** de eşleştirebilir.

**Temel yapı:**

```
match ifade:
    case değer1:
        işlem1
    case değer2:
        işlem2
    case _:
        varsayılan_işlem
```

_ karakteri, *her şeye uyan* yani “else” gibi çalışan bir yapıdır.

**Örnek 1 – Basit kullanım:**

```
gun = "pazartesi"

match gun:
    case "pazartesi":
        print("Haftanın ilk günü")
    case "cuma":
        print("Haftanın son iş günü")
    case "cumartesi" | "pazar":
        print("Hafta sonu")
    case _:
        print("Geçersiz gün")
```
```

Burada "cumartesi" | "pazar" ile birden fazla değeri aynı durumda yakalayabiliyoruz.

Örnek 2 – Sayılarla:

```
sayi = int(input("Bir sayı gir: "))

match sayi:
    case 0:
        print("Sıfır")
    case 1 | 2 | 3:
        print("Küçük sayı")
    case _:
        print("Büyük sayı")
```

Örnek 3 – Desen eşleme (pattern matching):

```
veri = ("kullanıcı", 25)

match veri:
    case ("kullanıcı", yas) if yas >= 18:
        print("Erişkin kullanıcı")
    case ("kullanıcı", yas):
        print("Reşit değil")
    case _:
        print("Bilinmeyen veri")
```

Burada yas değişkeni doğrudan desenden alınır ve koşul içinde kullanılabilir.

Özetle:
match-case, if–elif–else zincirlerini daha okunabilir hale getirir.
Basit karşılaştırmalardan karmaşık veri desenlerine kadar kullanılabilir, özellikle veri yapılarıyla (tuple, dict, list) çalışırken çok faydalıdır.

Not;

- Koşullu ifadeleri anlamakta zorlandıysanız Youtube, Google gibi platformlardan çeşitli örneklere bakmanız iyi olur.

5.Ünite - 1. Döngüler

Bir işlemi belirli bir koşul sağlandıkça tekrar eden yapılardır.
Python'da iki temel döngü vardır:

- **for döngüsü**
- **while döngüsü**

---

2. for Döngüsü

Liste, string, tuple gibi üzerinde gezilebilen yapılarda kullanılır.

```
for eleman in koleksiyon:
    işlem
```

Örnek:

```
for i in [1,2,3]:
    print(i)
```

---

3. range() Fonksiyonu

Sayı üretmek için kullanılır.

```
range(bitis)
range(baslangic, bitis)
range(bas, bitis, artis)
```

Örnek:

```
for i in range(0,10,2):
    print(i)
```

---

4. İç İçe Döngüler

Bir döngü içinde başka bir döngü.

```
for i in range(3):
    for j in range(2):
        print(i, j)
```

---

5. while Döngüsü

Koşul True olduğu sürece çalışır.

```
while koşul:
    işlem
```

Örnek:

```
sayi = 0
while sayi < 5:
    print(sayi)
    sayi += 1
```

---

6. break ve continue

- **break:** Döngüyü tamamen bitirir.
- **continue:** O turu geçer, döngü devam eder.

Örnek:

```
a = [2, 27, 35, 46, 39, 88, 100, 11]

i = 0
while i < len(a):
    if a[i] % 2 == 0:
        i += 1
        continue
    else:
        print(a[i], end=" ")
        i += 1
```

output:

```
27 35 39 11
```

---

7. Liste Kavramı (List)

Değiştirilebilir veri yapılarını tutar.

```
liste = [1,2,3]
liste.append(4)
```

---

8. List Comprehension

Kısa, temiz liste üretme yöntemi.

```
kares = [i*i for i in range(10)]
```

Koşullu kullanım:

```
ciftler = [i for i in range(20) if i % 2 == 0]
```

---

9. Fonksiyonlara Giriş

Fonksiyon, tekrar eden işlemleri toplar.

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

Parametre alabilir, değer döndürebilir.

6.Ünite - Fonksiyon Nedir?

Belirli bir işi yapan, tekrar kullanılabilir kod parçacıklarıdır.
Avantajları: modülerlik, okunabilirlik, hata tespiti kolaylığı.

- **Parametre:** Fonksiyonun aldığı girdi.
- **Argüman:** Fonksiyon çağrılırken verilen değer.
- **return:** Değer döndürür.

---

1. def ile Fonksiyon Tanımlama

Genel yapı:

```
def fonk_adi(parametreler):
    işlemler
    return değer
```

Örnek:

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

Parametreli:

```
def adaGoreSelamVer(isim):
    print(isim, "Merhaba!")
```

Birden fazla parametre:

```
def adSoyad_SelamVer(isim, soyisim):
    print(isim, soyisim, "Merhaba!")
```

---

2. Return ile Değer Döndürme

Tek değer:

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

Birden fazla değer (tuple):

```
def toplam_fark(a, b):
    return a+b, a-b
```

---

3. Varsayılan Parametre

```
def ciftSayilariGetir(n, bas=0):
    sayilar = []
    while len(sayilar) < n:
        if bas % 2 == 0:
            sayilar.append(bas)
        bas += 1
    return(sayilar)
```

Çağrı:

```
ciftSayilariGetir(5) # [0, 2, 4, 6, 8]
ciftSayilariGetir(5, 20) # [20, 22, 24, 26, 28]
```

---

4. lambda Fonksiyonları

Tek satırlık fonksiyonlar.

Örnek:

```
toplam = lambda a,b: a+b
tek_mi = lambda a: a % 2 == 1
```

---

5. map() ve filter()

```
liste = [0,1,2,3,4,5]

sonuc = map(lambda x: x*10, liste)
filtre = filter(lambda x: x % 2 == 1, liste)
```

---

6. Örnek Fonksiyon Problemleri

Asal sayı testi:

```
def asal_mi(n):
    if n >= 2:
        for i in range(2, n):
            if sayi % i == 0:
                return False
                break
            else:
                return True
    else:
        return False
```

Sayı yer değiştirme:

```
def degistir(a,b):
    return b,a
```

Metni ters çevirme:

```
def tersCevir(metin):
    return metin[::-1]
```

Sesli harf sayısı:

```
def sesliHarfSayisi(metin):
    ...
```

Tekrarlayan karakterleri kaldırma:

```
def tekrarlari_kaldir(metin):
    ...
```

Fibonacci listesi:

```
def fibonacciSerisi(n):
    ...
```

Liste eleman tipi kontrolü ve toplam:

```
def listeKontrol(liste):
    ...
```

Çember alan ve çevre (varsayılan pi):

```
def hesapla(r, pi=3.14):
    return 2*pi*r, pi*r*r
```

İstersen bu iki özetin birleşik tek bir PDF veya tek bir uzun MD halinde versiyonunu da hazırlayabilirim.

7.Ünite - Nesneye Yönelimli Programlama

- Nesneye Yönelimli Programlama, bir programı **özellikler** ve **davranışlar** içeren nesneler etrafında düzenleyen programlama yaklaşımıdır. Java, C#, Python, PHP ve JavaScript gibi birçok modern dil bu paradigmayı destekler.

- Gerçek dünyadaki varlıkların programlama ortamında temsil edilmesi mantığına dayanır. Örneğin bir **kişi** nesnesi; *ad, yaş* gibi özelliklere ve *yürüme, konuşma* gibi davranışlara sahip olabilir. Bir **e-posta** nesnesi de; *alıcı, konu, gövde* gibi özellikleri ve *gönderme, ek ekleme* gibi davranışları içerir.
- Prosedürel programlamada kod adım adım ilerlerken, OOP'de kod nesneler ve onların etkileşimleri etrafında şekillenir.

OOP'nin iki temel yapı taşı vardır:

1. Kalıtım (Inheritance)
Bir sınıfın başka bir sınıfın özellik ve davranışlarını miras almasıdır.
Örneğin *pizza yapan robotlar*, "robot" sınıfının genel özelliklerini otomatik olarak kazanır. Genel davranışlar bir kez yazılır, tüm alt türler tekrar kullanır.

2. Kompozisyon (Composition)
Bir nesnenin başka nesnelerden oluşmasıdır.
Pizza yapan bir robot; *kol, motor, sensör* gibi bileşenlerden oluşur. Her bileşen kendi sınıfıdır ve bir araya geldiklerinde robotun davranışını oluştururlar.

Özetle OOP, kodu daha modüler, yeniden kullanılabilir ve gerçek dünyaya daha yakın bir yapıda kurgulama tekniğidir.

Sınıf & Nesne

- `class SınıfAdı:` ile tanımlanır.
- **Özellik (attribute/state)**: nesnenin verileri.
- **Metot (method/behaviour)**: nesnenin eylemleri.
- `__init__(self, ...)` = yapıcı (constructor).
- Örnek:

```
class Evcil:
    tur = "Kedi"
    def __init__(self, ad):
        self.ad = ad
    def kısaBilgi(self):
        print(self.tur, self.ad)
```

Kompozisyon vs Kalıtım

- **Kompozisyon:** Nesne, diğer nesneleri içerir (parça-para whole).
- **Kalıtım (Inheritance):** `class Child(Parent):` — parent özellikleri child tarafından kullanılır/özelleştirilir.
- Parent `__init__` child içinde çağrılmalıdır (veya `super()` kullanılabilir).

Polimorfizm (Çok Biçimlilik)

- Aynı isimli metodun alt sınıflarda farklı davranması (method overriding).
- Örnek: `Sekil` sınıfı; `Kare` ve `Dikdortgen` kendi `alan()` / `cevre()` hesaplarını yapar.

Kapsülleme (Encapsulation)

- Veriyi gizleme; Python'da **özel** öznitelik için çift alt çizgi `__isim` kullanılır (isim kırpma — name mangling).
- Erişim/set için `get_...` / `set_...` metodları; setter içinde doğrulama yapılır (ör. puan 1–10 arası).

Örnek sınıflar

- `evcilHayvanPasaportu` (özellikler, metot, örnekleme)
- `Market` (urun_ekle, urun_sil, urunleri_listele)
- `Agac`, `MeyveAgaci`, `YaprakDokmeyenAgac` (kalıtım ve genişletme)
- `Calisan`, `Sinav`, `Film`, `Hasta` — kapsülleme ve getter/setter kullanımı.

Ozet

- Sınıfları modüler, tek sorumluluklu tut.
- __init__ içinde gerekli başlangıçları yap.
- Kapsülleme ile veri tutarlılığını koru (setter doğrulaması).
- Kalıtımı gerektiğinde, kompozisyonu tercih et, yeniden kullanım ve esneklik için.

8.Ünite - Python Numpy ve Pandas

---

1. NumPy (Numerical Python)

Çok boyutlu dizileri saklamak, vektör ve matrisler üzerinde hızlı matematiksel işlemler yapmak için kullanılır.

Kurulum ve Çağırma:

```
pip install numpy  # Kurulum
import numpy as np
```

Temel Dizi (Array) İşlemleri

- **Dizi Oluşturma:**
  - np.array([1, 2, 3]): Listeden dizi oluşturur.
  - np.zeros((2,2)): Sıfırlardan oluşan matris.
  - np.ones((2,3)): Birlerden oluşan matris.
  - np.arange(15): Belirli aralıkta sayı dizisi üretir.
  - np.random.randint(0, 10, 5) : Rastgele tam sayı üretir.
- **Dizi Özellikleri:**
  - .ndim: Boyut sayısı.
  - .shape: Boyut bilgisi (satır, sütun)
  - .size: Toplam eleman sayısı.
  - .dtype: Veri tipi.

Manipülasyon ve Hesaplama

- **İndeksleme & Dilimleme:**
  - A[0:5]: İlk 5 eleman.
  - A[::-1]: Diziyi ters çevirir.
- **Şekil Değiştirme:** np.reshape(D, (4,5)) dizinin boyutlarını değiştirir.
- **Birleştirme:** np.concatenate([a, b], axis=0) dizileri birleştirir.
- **İstatistiksel İşlemler:** np.sum(), np.mean(), np.max(), np.std()
- **Filtreleme:** A[A > 50] koşulu sağlayan elemanları getirir.

---

2. pandas Kütüphanesi

NumPy üzerine inşa edilmiştir. Veri analizi, temizleme ve işleme için **Series** ve **DataFrame** yapılarını sunar.

Kurulum ve Çağırma:

```
pip install pandas  # Kurulum
```

import pandas as pd

Veri Yapıları

1. **Series (Seriler):** Tek boyutlu, etiketli dizilerdir.
    - pd.Series([10, 20], index=["a", "b"]) şeklinde oluşturulur.
2. **DataFrame:** Satır ve sütunlardan oluşan iki boyutlu veri yapısıdır (Excel tablosu gibi).
    - Sözlükten oluşturma: pd.DataFrame({"Urun": ["Elma", "Armut"], "Fiyat": [10, 20]})

Dosya İşlemleri (I/O)

Veri setlerini okumak ve kaydetmek için kullanılır.

- **Okuma:**
    - pd.read_csv("dosya.csv"): CSV okuma

- pd.read_excel("dosya.xlsx"): Excel okuma (openpyxl gerektirir)

- **Yazma:**
- df.to_csv("yeni.csv") veya df.to_excel("yeni.xlsx")

Veri Keşfi ve Analizi

Veri setini hızlıca tanımak için kullanılan komutlar:

- df.head(5): İlk 5 satırı gösterir.
- df.tail(): Son satırları gösterir.
- df.info(): Bellek kullanımı ve veri tipleri hakkında özet bilgi.
- df.describe(): Sayısal sütunların istatistiksel özeti (ortalama, std, min, max)
- df["sütun"].value_counts(): Kategorik verilerin frekansını sayar

Veri Seçimi: loc ve iloc

DataFrame içerisinden satır/sütun seçmek için iki temel yöntem vardır:

Özellik	loc	iloc
Mantık	Etiket tabanlı (İsimle seçim)	İndeks tabanlı (Sayı ile seçim)
Aralık	Son eleman **dahil** ("A":"C")	Son eleman **hariç** (0:3)
Örnek	df.loc[:, "Ad"] (Tüm satırlar, "Ad" sütunu)	df.iloc[:, 0] (Tüm satırlar, 0. sütun)

Filtreleme Örneği:

df.loc[df["yas"] > 18] komutu, "yas" sütunu 18'den büyük olan satırları getirir.