Listeler, Demetler, Sözlükler ve Kümeler

Table of contents

Bölüm 5: Veri Yapıları - Listeler, Demetler, Sözlükler ve Kümeler

5.1 Veri Yapıları: Verileri Organize Etme

Veri yapıları, verileri bilgisayarın belleğinde düzenli bir şekilde saklamak ve işlemek için kullanılan yapılardır. Farklı veri tiplerini bir araya getirerek, daha karmaşık bilgileri temsil etmemizi ve bu bilgiler üzerinde işlemler yapmamızı sağlarlar.

Python, listeler, demetler, sözlükler ve kümeler gibi çeşitli yerleşik veri yapıları sunar. Her veri yapısının kendine özgü özellikleri ve kullanım alanları vardır.

5.2 Listeler (Lists)

Listeler, sıralı, değiştirilebilir (mutable) ve birden fazla veri tipini içerebilen koleksiyonlardır. Köşeli parantez (`[]`) kullanılarak tanımlanırlar ve elemanlar virgülle (`,`) ayrılır.

Örnekler:

```
sayilar = [1, 5, 3, 9, 7]
isimler = ["Ahmet", "Mehmet", "Ayşe"]
karisik_liste = [10, "Elma", True, 3.14]
bos_liste = []
```

5.2.1 Liste İndeksleme ve Dilimleme

Listelerdeki elemanlara, stringlerdeki gibi indekslerini kullanarak erişebiliriz. İndeksler 0'dan başlar.

Örnekler:

```
sayilar = [1, 5, 3, 9, 7]

print(sayilar[0])  # Çıktı: 1 (İlk eleman)
print(sayilar[2])  # Çıktı: 3
print(sayilar[-1]) # Çıktı: 7 (Son eleman)

print(sayilar[1:4])  # Çıktı: [5, 3, 9] (1. indeksten 4. indekse kadar (4 hariç))
```

```
print(sayilar[:3])   # Çıktı: [1, 5, 3] (Baştan 3. indekse kadar (3 hariç))
print(sayilar[2:])   # Çıktı: [3, 9, 7] (2. indeksten sona kadar)
```

5.2.2 Listelerde `in` ve `not in` Operatörleri

- `in`: Bir elemanın listede olup olmadığını kontrol eder.
- `not in`: Bir elemanın listede olup olmadığını kontrol eder.

```
meyveler = ["elma", "armut", "muz"]

print("elma" in meyveler)   # Çıktı: True
print("kivi" in meyveler)   # Çıktı: False
print("üzüm" not in meyveler) # Çıktı: True
```

5.2.3 Liste Metotları

Python, listeler üzerinde birçok işlem yapmak için kullanabileceğiniz çeşitli metotlar sunar.

- **`append(eleman)`:** Listenin sonuna yeni bir eleman ekler.

  ```
  sayilar = [1, 2, 3]
  sayilar.append(4)
  print(sayilar)  # Çıktı: [1, 2, 3, 4]

  renkler = ["kırmızı", "yeşil"]
  renkler.append("mavi")
  print(renkler) # Çıktı: ['kırmızı', 'yeşil', 'mavi']
  ```

- **`insert(indeks, eleman)`:** Belirtilen indekse yeni bir eleman ekler.

  ```
  isimler = ["Ahmet", "Mehmet"]
  isimler.insert(1, "Ayşe")
  print(isimler)  # Çıktı: ["Ahmet", "Ayşe", "Mehmet"]

  sayilar = [1, 3, 4]
  sayilar.insert(1, 2)
  print(sayilar) # Çıktı: [1, 2, 3, 4]
  ```

- **`remove(eleman)`:** İlk bulunan elemanı listeden siler.

- **Not:** Eğer eleman listede yoksa, `ValueError` hatası verir.

```
sayilar = [1, 2, 3, 2]
sayilar.remove(2)
print(sayilar)  # Çıktı: [1, 3, 2]

harfler = ["a", "b", "c", "a"]
harfler.remove("a")
print(harfler) # Çıktı: ['b', 'c', 'a']

# harfler.remove("d") # ValueError: list.remove(x): x not in list
```

- **`pop(indeks)`:** Belirtilen indeksteki elemanı listeden çıkarır ve döndürür.
- İndeks belirtilmezse son elemanı çıkarır.
- **Not:** Eğer indeks geçersizse (listenin sınırları dışında ise), `IndexError` hatası verir.

```
sayilar = [1, 2, 3]
silinen = sayilar.pop(1)
print(silinen)  # Çıktı: 2
print(sayilar)  # Çıktı: [1, 3]

meyveler = ["elma", "armut", "muz"]
son_meyve = meyveler.pop()
print(son_meyve) # Çıktı: muz
print(meyveler) # Çıktı: ['elma', 'armut']

# sayilar.pop(5) # IndexError: pop index out of range
```

- **`index(eleman)`:** Belirtilen elemanın ilk bulunduğu indeksi döndürür.
- **Not:** Eğer eleman listede yoksa, `ValueError` hatası verir.

```
meyveler = ["elma", "armut", "muz"]
indeks = meyveler.index("armut")
print(indeks)  # Çıktı: 1

sayilar = [10, 20, 30, 40]
print(sayilar.index(30)) # Çıktı: 2

# print(sayilar.index(50)) # ValueError: 50 is not in list
```

- **`count(eleman)`:** Belirtilen elemanın listede kaç kez geçtiğini döndürür.

```
sayilar = [1, 2, 3, 2, 2, 4]
adet = sayilar.count(2)
print(adet)  # Çıktı: 3

harfler = ['a', 'b', 'c', 'd', 'a', 'a', 'b']
print(harfler.count('a')) # Çıktı: 3
```

- **`sort()`:** Listeyi artan sırada sıralar.
- **Not:** `sort()` metodu listeyi kalıcı olarak değiştirir.
- `reverse=True` parametresi ile azalan sırada sıralanabilir.

```
sayilar = [3, 1, 4, 2]
sayilar.sort()
print(sayilar)  # Çıktı: [1, 2, 3, 4]

isimler = ["Zehra", "Ali", "Can"]
isimler.sort()
print(isimler) # Çıktı: ['Ali', 'Can', 'Zehra']

sayilar = [3, 1, 4, 2]
sayilar.sort(reverse=True)
print(sayilar) # Çıktı: [4, 3, 2, 1]
```

- **`reverse()`:** Listenin elemanlarını tersine çevirir.
- **Not:** `reverse()` metodu da listeyi kalıcı olarak değiştirir.

```
sayilar = [1, 2, 3, 4]
sayilar.reverse()
print(sayilar)  # Çıktı: [4, 3, 2, 1]

kelimeler = ["Python", "Java", "C++"]
kelimeler.reverse()
print(kelimeler) # Çıktı: ['C++', 'Java', 'Python']
```

5.2.4 Listelerle Döngüler: Tek Tek Elemanlara Erişme

```
sayilar = [1, 2, 3, 4, 5]

for sayi in sayilar:
    print(sayi * 2, end=" ")  # Çıktı: 2 4 6 8 10
```

```
isimler = ["Ali", "Veli", "Ayşe"]
for isim in isimler:
    print("Merhaba", isim)
```

5.2.5 Listelerin Kopyalanması: Dikkat!

- Listeler değiştirilebilir (mutable) veri tipleridir.
- Bir listeyi başka bir değişkene atarken, sadece referansı kopyalanır, elemanlar değil.
- `copy()` metodu veya dilimleme ile listenin bir kopyasını oluşturabiliriz.

```
# Yanlış Kopyalama:
liste1 = [1, 2, 3]
liste2 = liste1

liste2.append(4)

print(liste1)  # Çıktı: [1, 2, 3, 4]
print(liste2)  # Çıktı: [1, 2, 3, 4]
```

```
# Doğru Kopyalama (copy() metodu):
liste1 = [1, 2, 3]
liste2 = liste1.copy()

liste2.append(4)

print(liste1)  # Çıktı: [1, 2, 3]
print(liste2)  # Çıktı: [1, 2, 3, 4]
```

```
# Doğru Kopyalama (dilimleme):
liste1 = [1, 2, 3]
liste2 = liste1[:]

liste2.append(4)

print(liste1)  # Çıktı: [1, 2, 3]
print(liste2)  # Çıktı: [1, 2, 3, 4]
```

5.2.6 `join()` Metodu: Liste Elemanlarını Birleştirme

- `join()` metodu, bir string'i ayraç olarak kullanarak listedeki string elemanlarını birleştirir ve tek bir string döndürür.

```
kelimeler = ["Python", "eğlenceli", "bir", "dildir"]
cumle = " ".join(kelimeler)
print(cumle)  # Çıktı: Python eğlenceli bir dildir

harfler = ["a", "b", "c"]
yeni_metin = "-".join(harfler)
print(yeni_metin) # Çıktı: a-b-c
```

5.2.7 `extend()` Metodu: Listeleri Birleştirme

- `extend()` metodu, bir listenin sonuna başka bir listenin elemanlarını ekler.

```
liste1 = [1, 2, 3]
liste2 = [4, 5, 6]

liste1.extend(liste2)
print(liste1)  # Çıktı: [1, 2, 3, 4, 5, 6]

meyveler1 = ["elma", "armut"]
meyveler2 = ["muz", "çilek"]
meyveler1.extend(meyveler2)
print(meyveler1) # Çıktı: ['elma', 'armut', 'muz', 'çilek']
```

5.2.8 Listeleri Toplama (+)

- + operatörü ile iki listeyi birleştirebiliriz.
- Bu işlem yeni bir liste oluşturur, orijinal listeler değişmez.

```
liste1 = [1, 2, 3]
liste2 = [4, 5, 6]

liste3 = liste1 + liste2
print(liste3)  # Çıktı: [1, 2, 3, 4, 5, 6]

harfler1 = ['a', 'b', 'c']
```

```
harfler2 = ['d', 'e', 'f']
print(harfler1 + harfler2) # Çıktı: ['a', 'b', 'c', 'd', 'e', 'f']
```

5.2.9 Dilim Değiştirme

- Listelerin belirli bir bölümünü (dilimini) yeni değerlerle değiştirebiliriz.

```
sayilar = [10, 20, 30, 40, 50, 60]
sayilar[1:3] = [11, 61] # 2. ve 3. elemanları değiştir
print(sayilar) # Çıktı: [10, 11, 61, 40, 50, 60]

sayilar[1:6] = [12, 62] # 2. elemandan 6. elemana kadar olanları 2 yeni elemanla değiştir
print(sayilar) # Çıktı: [10, 12, 62]
```

5.3 Demetler (Tuples)

Demetler, sıralı, değiştirilemez (immutable) ve birden fazla veri tipini içerebilen koleksiyonlardır. Parantez (`()`) kullanılarak tanımlanırlar ve elemanlar virgülle (`,`) ayrılır.

Örnek:

- **Koordinatlar:** `koordinat = (10, 20)`
- **RGB Renk Kodu:** `renk = (255, 0, 0)` (Kırmızı)

```
gunler = ("Pazartesi", "Salı", "Çarşamba")
```

5.3.1 Demetlerde `in` ve `not in` Operatörleri

- `in`: Bir elemanın demette olup olmadığını kontrol eder.
- `not in`: Bir elemanın demette olup olmadığını kontrol eder.

```
gunler = ("Pazartesi", "Salı", "Çarşamba")

print("Salı" in gunler)  # Çıktı: True
print("Pazar" in gunler) # Çıktı: False
print("Cuma" not in gunler) # Çıktı: True
```

5.3.2 Demet İndeksleme ve Dilimleme

- Listeler gibi indekslenir ve dilimlenirler.

```
koordinatlar = (10, 20, 30)

print(koordinatlar[0])  # Çıktı: 10
print(koordinatlar[1:]) # Çıktı: (20, 30)

harfler = ('a', 'b', 'c', 'd', 'e')
print(harfler[2:4]) # Çıktı: ('c', 'd')
```

5.3.3 Demet Metotları

- Demetler değiştirilemez oldukları için sınırlı sayıda metoda sahiptirler.
- `count(eleman)`: Belirtilen elemanın demette kaç kez geçtiğini döndürür.

  ```
  renkler = ("kırmızı", "yeşil", "mavi", "kırmızı")
  adet = renkler.count("kırmızı")  # adet değişkeninin değeri 2 olur
  print(adet) # Çıktı: 2
  ```

- `index(eleman)`: Belirtilen elemanın demetteki ilk bulunduğu indeksi döndürür. Eğer eleman demette yoksa, `ValueError` hatası verir.

  ```
  renkler = ("kırmızı", "yeşil", "mavi")
  indeks = renkler.index("yeşil")  # indeks değişkeninin değeri 1 olur
  print(indeks) # Çıktı: 1
  ```

- **Ek Demet Metotları:**
  - `len(demet)`: Demetin uzunluğunu (eleman sayısını) döndürür.
  - `max(demet)`: Demetin en büyük elemanını döndürür (elemanlar karşılaştırılabilir olmalıdır).
  - `min(demet)`: Demetin en küçük elemanını döndürür (elemanlar karşılaştırılabilir olmalıdır).

```
sayilar = (3, 1, 4, 1, 5, 9, 2, 6, 5)
print(len(sayilar)) # Çıktı: 9
print(max(sayilar)) # Çıktı: 9
print(min(sayilar)) # Çıktı: 1
```

5.3.4 Demetlerin Avantajları

- **Değiştirilemezlik:** Demetlerin değiştirilemez olması, verilerin yanlışlıkla değiştirilmesini önler ve program güvenliğini artırır.
- **Performans:** Demetler, listelerden daha hızlı erişim sağlar, çünkü elemanları sabittir.
- **Anahtar Olarak Kullanım:** Demetler, sözlüklerde anahtar olarak kullanılabilir, çünkü değiştirilemez olmaları gerekir.

5.4 Sözlükler (Dictionaries)

Sözlükler, anahtar-değer (key-value) çiftleri şeklinde verileri saklayan, değiştirilebilir (mutable) ve sırasız koleksiyonlardır. Süslü parantez (`{}`) kullanılarak tanımlanırlar ve anahtarlar ve değerler iki nokta üst üste (`:`) ile ayrılır. Elemanlara anahtarları kullanılarak erişilir.

Örnek:

- **Telefon Rehberi:** `telefon_rehberi = {"Ahmet": "555-1234", "Mehmet": "555-5678"}`
- **Ürün Kataloğu:** `ürünler = {"kalem": 2.50, "defter": 5.00, "silgi": 1.00}`

```
ogrenci = {"ad": "Ahmet", "soyad": "Yılmaz", "yas": 20}
```

5.4.1 Sözlüklerde `in` ve `not in` Operatörleri

- `in`: Bir anahtarın sözlükte olup olmadığını kontrol eder.
- `not in`: Bir anahtarın sözlükte olup olmadığını kontrol eder.

Not: `in` operatörü, sözlüklerde **sadece anahtarları** kontrol eder.

```
ogrenci = {"ad": "Ahmet", "soyad": "Yılmaz", "yas": 20}

print("ad" in ogrenci)     # Çıktı: True
print("bolum" in ogrenci)  # Çıktı: False
print("yas" not in ogrenci) # Çıktı: False
```

5.4.2 Sözlük Elemanlarına Erişme

- Anahtarlar kullanılarak elemanlara erişebiliriz.

```
ogrenci = {"ad": "Ahmet", "soyad": "Yılmaz", "yas": 20}

print(ogrenci["ad"])    # Çıktı: Ahmet
print(ogrenci["yas"])   # Çıktı: 20

print(urun["fiyat"])  # Çıktı: 5.50
```

5.4.3 Sözlüklerde Eleman Ekleme, Değiştirme ve Silme

- **Yeni eleman ekleme:**

```
ogrenci["bolum"] = "Bilgisayar Mühendisliği"
print(ogrenci) # Çıktı: {'ad': 'Ahmet', 'soyad': 'Yılmaz', 'yas': 20, 'bolum': 'Bilgisayar Mü
```

- **Varolan bir elemanın değerini değiştirme:**

```
ogrenci["yas"] = 21
print(ogrenci) # Çıktı: {'ad': 'Ahmet', 'soyad': 'Yılmaz', 'yas': 21, 'bolum': 'Bilgisayar Mü
```

- **Bir elemanı silme:**

```
del ogrenci["soyad"]
print(ogrenci) # Çıktı: {'ad': 'Ahmet', 'yas': 21, 'bolum': 'Bilgisayar Mühendisliği'}
```

5.4.4 Sözlük Metotları

- `keys()`: Sözlükteki tüm anahtarları döndürür.
- `values()`: Sözlükteki tüm değerleri döndürür.
- `items()`: Sözlükteki tüm anahtar-değer çiftlerini döndürür.
- `get(anahtar, varsayılan_değer)`: Belirtilen anahtarın değerini döndürür. Anahtar yoksa, varsayılan değeri döndürür.
- `pop(anahtar, varsayılan_değer)`: Belirtilen anahtara karşılık gelen değeri sözlükten çıkarır ve döndürür. Anahtar yoksa, varsayılan değeri döndürür.
- `update(diger_sozluk)`: Sözlüğü, başka bir sözlükteki anahtar-değer çiftleriyle günceller.

```
ogrenci = {"ad": "Ahmet", "soyad": "Yılmaz", "yas": 20}

print(ogrenci.keys())   # Çıktı: dict_keys(['ad', 'soyad', 'yas'])
print(ogrenci.values()) # Çıktı: dict_values(['Ahmet', 'Yılmaz', 20])
print(ogrenci.items())  # Çıktı: dict_items([('ad', 'Ahmet'), ('soyad', 'Yılmaz'), ('yas', 20

print(urun.keys()) # Çıktı: dict_keys(['isim', 'fiyat', 'stok'])

print(ogrenci.get("bolum", "Bilgi Yok")) # Çıktı: Bilgi Yok
print(ogrenci.pop("yas", 0)) # Çıktı: 20
print(ogrenci) # Çıktı: {'ad': 'Ahmet', 'soyad': 'Yılmaz'}

ogrenci.update({"bolum": "Bilgisayar Mühendisliği", "not_ortalamasi": 3.5})
print(ogrenci) # Çıktı: {'ad': 'Ahmet', 'soyad': 'Yılmaz', 'bolum': 'Bilgisayar Mühendisliği
```

- **popitem():** Python 3.7 ve sonrasında son girilen öğeyi döner ve siler. Boş sözlükte `KeyError` hatası verir.

```
stok = {
    "kalem": 100,
    "defter": 50,
    "silgi": 75
    }
stok.popitem() # ('silgi', 75)
print(stok) # Çıktı: {'kalem': 100, 'defter': 50}
```

- **setdefault():** `setdefault()` metodu, bir sözlükte belirli bir anahtarı kontrol edip, bu anahtar mevcutsa ilgili değeri döner, yoksa sözlüğe bu anahtarı ekleyip ona bir varsayılan değer atar.

```
bilgiler = {"isim": "Ahmet", "yas": 30}
sonuc = bilgiler.setdefault("isim", "Mehmet")
print(sonuc)     # Çıktı: Ahmet
print(bilgiler)    # Çıktı: {"isim": "Ahmet", "yas": 30}

sonuc = bilgiler.setdefault("meslek", "Mühendis")
print(sonuc)       # Çıktı: Mühendis
print(bilgiler)   # Çıktı: {'isim': 'Ahmet', 'yas': 30, 'meslek': 'Mühendis'}
```

- **fromkeys():** Verilen bir iterable'dan yeni bir sözlük oluşturmak için kullanılır.

```
anahtarlar = ['isim', 'yas', 'meslek']
sozluk = dict.fromkeys(anahtarlar)
print(sozluk) # Çıktı: {'isim': None, 'yas': None, 'meslek': None}

degerler = ["Bilinmiyor"] * len(anahtarlar)
sozluk2 = dict.fromkeys(anahtarlar, degerler)
print(sozluk2) # Çıktı: {'isim': ['Bilinmiyor', 'Bilinmiyor', 'Bilinmiyor'], 'yas': ['Bilinm
```

- **`copy()`:** Verilen bir sözlüğü kopyalar.
- **`clear()`:** Verilen bir sözlüğü temizler.

```
user = {"isim": "Ahmet", "yas": 30}
user2 = user.copy()
print(user) # Çıktı: {'isim': 'Ahmet', 'yas': 30}
print(user2) # Çıktı: {'isim': 'Ahmet', 'yas': 30}

user.clear()
print(user) # Çıktı: {}
print(user2) # Çıktı: {'isim': 'Ahmet', 'yas': 30}
```

5.4.5 Sözlüklerde Döngüler

- Anahtarlar veya anahtar-değer çiftleri üzerinde döngü kullanabiliriz.

```
ogrenci = {"ad": "Ahmet", "soyad": "Yılmaz", "yas": 20}

for anahtar in ogrenci:
    print(anahtar, ":", ogrenci[anahtar])

for anahtar, deger in urun.items():
    print(f"{anahtar}: {deger}")
```

5.5 Kümeler (Sets)

Kümeler, benzersiz (unique) elemanlardan oluşan, değiştirilebilir (mutable) ve sırasız koleksiyonlardır. Süslü parantez (`{}`) ile tanımlanırlar veya `set()` fonksiyonu kullanılarak oluşturulabilirler.

Örnek:

- **Tekrar Eden Kelimeleri Bulma:** Bir metindeki tekrar eden kelimeleri bulmak için bir küme kullanabiliriz.
- **Kullanıcı İzinleri:** Bir kullanıcının bir web sitesinde veya uygulamada sahip olduğu izinleri saklamak için bir küme kullanabiliriz.

```
meyveler = {"elma", "armut", "muz"}
sayilar = set([1, 2, 3, 4, 5])
```

5.5.1 Kümelerde `in` ve `not in` Operatörleri

- `in`: Bir elemanın kümede olup olmadığını kontrol eder.
- `not in`: Bir elemanın kümede olup olmadığını kontrol eder.

```
meyveler = {"elma", "armut", "muz"}

print("elma" in meyveler)   # Çıktı: True
print("kivi" in meyveler)   # Çıktı: False
print("üzüm" not in meyveler) # Çıktı: True
```

5.5.2 Küme Metotları

- `add(eleman)`: Kümeye yeni bir eleman ekler (zaten varsa, hiçbir şey yapmaz).

```
meyveler = {"elma", "armut", "muz"}
meyveler.add("kivi")
print(meyveler) # {'kivi', 'muz', 'armut', 'elma'} (Sıra farklı olabilir)

sayilar = {1, 2, 3}
sayilar.add(4)
print(sayilar) # Çıktı: {1, 2, 3, 4} (Sıra farklı olabilir)
```

- `remove(eleman)`: Belirtilen elemanı kümeden siler (yoksa, `KeyError` hatası verir).

```
meyveler = {"elma", "armut", "muz"}
meyveler.remove("kivi")
print(meyveler) # {'elma', 'muz', 'armut'}
meyveler.remove("kavun") # KeyError: 'kavun'
```

- `discard(eleman)`: Belirtilen elemanı kümeden siler (yoksa, hata vermez).

```
meyveler = {"elma", "armut", "muz"}
meyveler.discard("kivi")
print(meyveler) # {'elma', 'muz', 'armut'}
meyveler.discard("kavun")
```

- `union(diger_kume)`: İki kümenin birleşimini döndürür.

```
kume1 = {1, 2, 3}
kume2 = {3, 4, 5}
kume3 = kume1.union(kume2)
print(kume1) # {1, 2, 3}
print(kume2) # {3, 4, 5}
print(kume3) # {1, 2, 3, 4, 5}
```

- `intersection(diger_kume)`: İki kümenin kesişimini döndürür.

```
kume1 = {1, 2, 3}
kume2 = {3, 4, 5}
print(kume1.intersection(kume2)) # {3}
```

- `difference(diger_kume)`: İlk kümenin, ikinci kümede olmayan elemanlarını içeren bir küme döndürür.

```
kume1 = {1, 2, 3}
kume2 = {3, 4, 5}
print(kume1.difference(kume2)) # {1, 2}
```

5.5.3 Kümelerle Döngüler

```
meyveler = {"elma", "armut", "muz"}

for meyve in meyveler:
    print(meyve)
```

5.6 Python'da Yerleşik Fonksiyonlar

Python'da yerleşik olarak bulunan birçok fonksiyon, veri yapıları ile çalışmayı ve matematiksel işlemler yapmak gibi işlemleri oldukça kolaylaştırır.

Bu fonksiyonlardan bazıları liste, tuple gibi dizilerin elemanlarını toplama, sıralama, minimum veya maksimum değerlerini bulma gibi işlemleri sağlar.

- **`sum()`:** Bir dizinin (liste, tuple gibi) sayısal elemanlarını toplar.
  - Parametre olarak başlangıç değeri verilebilir.

```
sayilar = [10, 20, 30, 40]
toplam = sum(sayilar)
toplam2 = sum(sayilar, 30)
print(toplam)  # Çıktı: 100
print(toplam2) # Çıktı: 130
```

- **`max()`:** Bir dizinin en büyük elemanını döndürür.

```
sayilar = [10, 20, 30, 40]
en_buyuk = max(sayilar)
print(en_buyuk)  # Çıktı: 40
print(max(3, 1, 5))  # Çıktı: 5
```

- **`min()`:** Bir dizinin en küçük elemanını döndürür.

```
sayilar = [10, 20, 30, 40]
en_kucuk = min(sayilar)
print(en_kucuk)      # Çıktı: 10
print(min(3, 1, 5))  # Çıktı: 1
```

- **`len()`:** Bir iterable'ın (liste, tuple, string vb.) eleman sayısını döndürür.

```
sayilar = [10, 20, 30, 40]
uzunluk = len(sayilar)
print(uzunluk)     # Çıktı: 4
kelime = "Python"
print(len(kelime)) # Çıktı: 6
```

- **`sorted()`:** Bir diziyi sıralı bir şekilde döndürür. Orijinal diziyi değiştirmez.

```
sayilar = [30, 10, 50, 20, 40]
sirali_sayilar = sorted(sayilar)
print(sirali_sayilar)  # Çıktı: [10, 20, 30, 40, 50]
ters_sirali_sayilar = sorted(sayilar, reverse=True)
print(ters_sirali_sayilar)  # Çıktı: [50, 40, 30, 20, 10]
```

- **`reversed()`:** Bir diziyi tersten sıralanmış şekilde döndürür. Ancak orijinal diziyi değiştirmez.

```
sayilar = [10, 20, 30, 40]
ters_sayilar = list(reversed(sayilar))
print(ters_sayilar)  # Çıktı: [40, 30, 20, 10]
```

- **`all()`:** Verilen dizideki tüm elemanlar `True` değerinde ise `True` döner. Aksi halde `False` döner.

```
degerler = [True, True, True]
print(all(degerler))  # Çıktı: True

degerler = [True, False, True]
print(all(degerler))  # Çıktı: False
```

- **`any()`:** Verilen dizide en az bir eleman `True` ise `True` döner.

```
degerler = [False, False, True]
print(any(degerler))  # Çıktı: True

degerler = [False, False, False]
print(any(degerler))  # Çıktı: False
```

- **`enumerate()`:** Bir iterable'ı numaralandırarak döner, her bir öğe ile birlikte indeksini verir.

```
meyveler = ["elma", "armut", "muz"]
for indeks, meyve in enumerate(meyveler):
    print(f"{indeks}: {meyve}")
```

```
0: elma
1: armut
2: muz
```

- **`zip()`:** Birden fazla iterable'ı (örneğin, listeler, demetler) aynı anda ele alarak bu iterable'ların her birinden sırayla bir eleman alıp, bu elemanları birleştirerek tuple'lar oluşturan bir fonksiyondur.

```
isimler = ["Ahmet", "Mehmet", "Ayşe"]
yaslar = [25, 30, 22]

# İki listeyi zip ile birleştir
birlesim = zip(isimler, yaslar)

# Sonuç olarak tuple'lar içeren bir zip nesnesi döner
print(list(birlesim))  # [('Ahmet', 25), ('Mehmet', 30), ('Ayşe', 22)]
```

5.7 Hangi Veri Yapısını Kullanmalıyım?

- **Sıralı ve değiştirilebilir bir koleksiyon, tekrarlayan elemanlara izin veriyorsanız:** Liste (`list`)
- **Sıralı ve değiştirilemez bir koleksiyon, veri bütünlüğü önemliyse ve daha hızlı erişime ihtiyaç duyuyorsanız:** Demet (`tuple`)
- **Anahtar-değer çiftleri ile veri saklamak, hızlı anahtar tabanlı erişim sağlamak için:** Sözlük (`dict`)
- **Benzersiz elemanlardan oluşan bir koleksiyon, küme işlemleri (birleşim, kesişim, fark) için:** Küme (`set`)

5.8 Bölüm Sonu Quiz

1. Aşağıdaki kod bloğunun çıktısı nedir?

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

2. Aşağıdaki kod bloğunun çıktısı nedir?

```
demet = (1, 2, 3, 4, 5)
print(demet[1:4])
```

3. Aşağıdaki kod bloğunun çıktısı nedir?

```
sayilar = [1, 5, 2, 4, 3]
sayilar.sort()
print(sayilar[0])
```

4. Aşağıdaki kod bloğunun çıktısı nedir?

```
meyveler = {"elma", "armut", "muz"}
meyveler.add("elma")
print(len(meyveler))
```

5. Aşağıdaki kod bloğunun çıktısı nedir?

```
kisi = {"ad": "Ali", "yas": 30}
print(kisi.get("meslek", "Bilinmiyor"))
```

6. Aşağıdaki kod bloğunun çıktısı nedir?

```
sayilar = [1, 2, 2, 3, 4, 4, 5]
benzersiz_sayilar = set(sayilar)
print(benzersiz_sayilar)
```

5.9 Alıştırmalar

1. Kullanıcıdan alınan 5 sayıyı bir listeye ekleyin.
2. Listedeki sayıların toplamını hesaplayın.
3. Listedeki en büyük sayıyı bulun.
4. Listedeki tüm çift sayıları yeni bir listeye ekleyin.
5. Kullanıcıdan alınan 3 meyve adını bir kümeye ekleyin.
6. Kümedeki elemanları alfabetik sıraya göre yazdırın.
7. Bir öğrenci not sistemi oluşturan bir program yazın. Program aşağıdaki özellikleri sağlamalıdır:
    - Kullanıcıdan öğrenci adı, soyadı ve 3 notu alın (vize1, vize2, final).
    - Kullanıcı ad bilgisine "q" değeri girene kadar yeni öğrenci bilgisi almaya devam edin.
    - Bu bilgileri bir sözlükte saklayın.
    - Her öğrenci için bir liste oluşturun ve notlarını bu listeye ekleyin.
    - Öğrencinin ortalama notunu hesaplayın (vize1: %30, vize2: %30, final: %40).
    - Öğrencinin harf notunu hesaplayın (90-100: AA, 80-89: BA, 70-79: BB, 60-69: CB, 0-59: FF).
    - Tüm öğrenci bilgilerini (ad, soyad, notlar, ortalama, harf notu) ekrana yazdırın.
8. Basit bir alışveriş sepeti uygulaması yazın. Program aşağıdaki özellikleri sağlamalıdır:
    - Ürünlerin adını ve fiyatını içeren bir sözlük oluşturun.
    - Kullanıcıdan ürün seçmesini ve miktar (tamsayı) girmesini isteyin.
    - Seçilen ürün ve miktarı bir listeye ekleyin.

- Var olan ürün tekrar eklenirse önceki miktara eklenir.
- Kullanıcı “tamam” yazana kadar ürün eklemeye devam etsin.
- Kullanıcı “tamam” yazdığında, toplam tutarı hesaplayın ve yazdırın.