Diziler

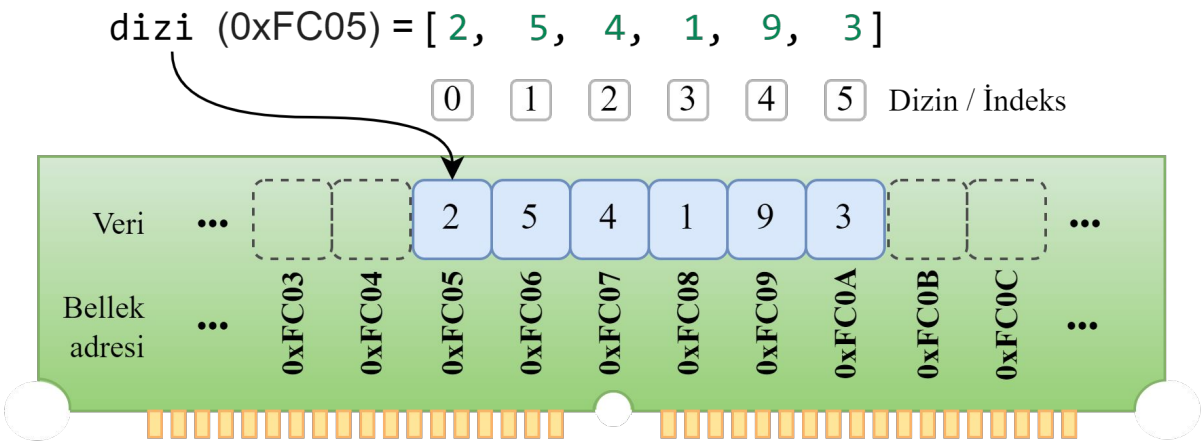
Dr. Hakan TEMİZ

Diziler

- Diziler, arrays , sequences adlarıyla da anılır. Pyhton'da, list , tuple ve str sınıfları, dizi tipinde olan hazır (built-in) veri yapılarıdır.
- Diziler, ardışık tutulan veri yapılarını ifade eder. Verinin elemanları birbiri ardına tutulur. Dizinin başlangıç konumu bilindiğinde, dizinin tüm elemanlarına başlangıçtan itibaren göreceli konum bilgisi (ki buna genellikle indeks/dizin diyoruz) ile erişmek mümkündür.
- Python dahil, çoğu programlama dilinde, indekslemeye 0 ile başlandığı için, dizinin ilk elemanının indeksi 0 olduğunu not edelim.
- Bir dizi (list) oluşturalım:

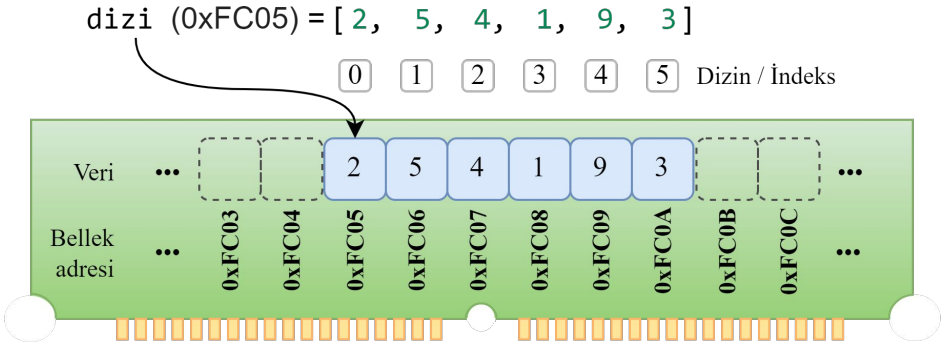
```
dizi = [2, 5, 4, 1, 9, 3] # bir python listesi
```

Diziler


dizi (0xFC05) = [2, 5, 4, 1, 9, 3]
0 1 2 3 4 5 Dizin / İndeks
Veri
Bellek adresi
2 5 4 1 9 3
0xFC03 0xFC04 0xFC05 0xFC06 0xFC07 0xFC08 0xFC09 0xFC0A 0xFC0B 0xFC0C

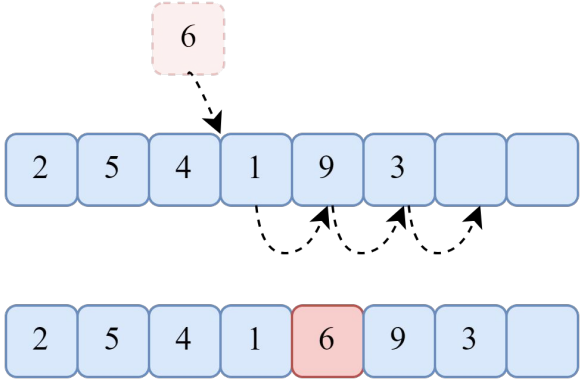
Diziler

- **dizi** değişkeni esasen bellekte bir adresin takma adıdır: 0xFC05
- İk elemanına (2) **dizi[0]** ifadesiyle erişilir; dizinin başlangıç adresidir: 0xFC05
- 0xFC06 adresindeki bir sonraki (5) elemana **dizi[1]** ifadesiyle erişilir.
- Dizinin tüm elemanlarına bu şekilde indekslemek suretiyle, bellekteki adresini bilmeye gerek kalmadan erişmek mümkündür.
- Son eleman (3), 0xFC0A adresindedir ve **dizi[5]** (veya **dizi[-1]**) indeksi ile erişilir.



Diziler - Ekleme

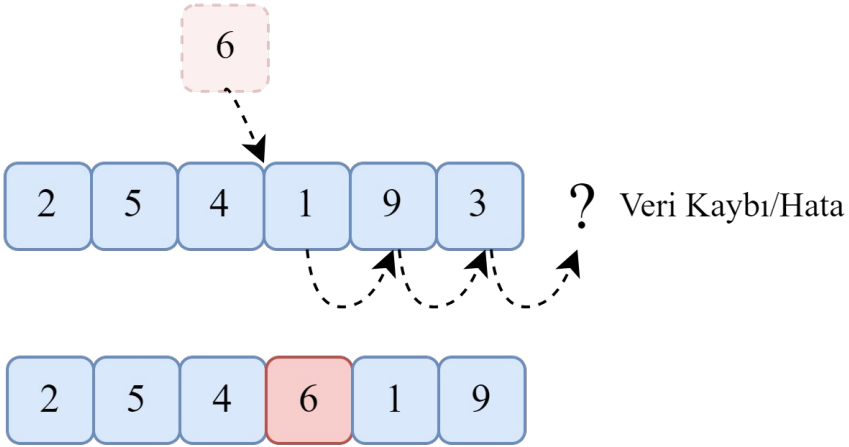
- Bir diziye eleman eklemek için, eklenecek dizinden sonraki tüm elemanların ötelenmesi gerekir. Aşağıdaki örnekte, 6 değerinin, 4 ile 1 arasına konabilmesi için 1, 9 ve 3 değerlerinin bir ileri konuma kayıt edilmesi gerekir.



- En kötü durum, dizinin en başına ekleme yapmaktır. Çünkü, bu durumda dizinin mevcut tüm öğelerinin bir ileri taşınması ki bu da O(N) işlem demektir

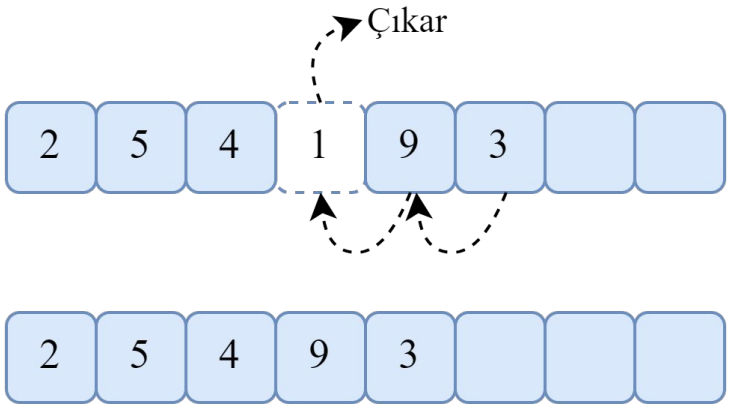
Diziler - Ekleme

- Tam dolu dizide eleman eklemek veri kaybı ve taşma sorunlarına yol açabilir. Boş bir yer olmadığından, son eleman kaybedilir veya program hata ile sonlanır.



Diziler - Silme

- Diziden bir eleman silmek için, silme konumundan sonraki tüm elemanlar birer sola kaydırılır; yani, bir önceki indeks konumlarına yeniden kayıt edilmeleri gerekir.
- Aşağıdaki örnekte 1 öğesinin silinmesi nedeniyle 9 ve 3 öğeleri birer sola kaydırılır.



- Silmede en kötü durum, dizideki ilk elemanın çıkarılmasıdır. Geriye kalan tüm elemanların birer sola (başa doğru) kaydırılması gerekir.

Diziler - Uygulama Alanları

- Diziler şu tür işlerde yaygın kullanılır:
  - Listeler, yığınlar, karma tablolar, vektörler ve matrisler gibi diğer veri yapılarının uygulanmasında,
  - Veritabanı kayıtlarının tutulmasında,
  - Bilgisayarda arama tablolarında.

Diziler - Avantajlar

- Çok basit bir veri yapısına sahiptir ve kolay tasarlanır ve uygulanır.
- Öğelere doğrudan O(1) işlemle erişilir (dizi[15] gibi). Öğelere erişim boyuttan tamamen bağımsızdır ve ciddi hız farkı yaratır.
- Verilerin bitişik bellek konumlarında depolanması veriye hızlı ulaşım sağlar.
- Aynı türdeki çoklu verileri tek bir ad ile temsil etmek dizilerle çok kolaydır.
- Boyut önceden bellidir ve genellikle blok alan tahsis edilir. Bu da, belleğin parçalı kullanılmasından kaynaklanabilecek sorunların azalmasına katkı sağlar.
- Diziler, her türlü (fakat tek bir tip) veri saklamak için kullanılabilir.
- Dizi veri yapısı çoğu donanım mimarisi ile kusursuz uyumludur.
- Bağlantılı liste, yığın, kuyruk, ağaç, graf vb. veri yapılarını uygulamada kullanılır.

Diziler - Dezavantajlar

- Diziler sabit bir boyuta sahiptir, yani statiktir. Bellek tahsis edildikten sonra artırılamaz veya azaltılamaz. Dolayısıyla ilave veri depolamaya elverişli değildir.
- Bir diziye gerekenden az bellek ayırmak, taşma hatasına ve veri kaybına yol açar.
- Gereğinden fazla bellek ayrılması, bellek israfına neden olur.
- Dizi yalnızca bir türde veri tipini saklar; farklı tipte değerleri depolayamaz.
- Verilerin bellekte ardışık konumlarda depolanması, özellikle aradan silme ve eklemeyi çok zorlaştırır. Bu sorun, öğelere ardışık olarak erişilmesine izin veren bağlı listelerin uygulanmasıyla aşılır.

Diziler - Karmaşıklık - Çalışma Zamanı

İşlem	En İyi Durum Ω	Ortalama Durum Θ	En Kötü Durum O
Gezinme	Ω(N)	Θ(N)	O(N)
Ekleme	Ω(1)	Θ(N)	O(N)
Silme	Ω(1)	Θ(N)	O(N)
Arama	Ω(1)	Θ(N)	O(N)

Diziler - Karmaşıklık - Bellek

İşlem	En İyi Durum Ω	Ortalama Durum Θ	En Kötü Durum O
Gezinme	Ω(N)	Θ(1)	O(1)
Ekleme	Ω(1)	Θ(N)	O(N)
Silme	Ω(1)	Θ(N)	O(N)
Arama	Ω(1)	Θ(1)	O(1)

Diziler - Python’da, C Dili Benzeri Veri Tipleri

Kod	C Veri Tipi	Kaç Bayt
'b'	işaretli karakter (signed char)	1
'B'	işaretsiz tamsayı (unsigned char)	1
'u'	unikod karakter (unicode char)	2 veya 4
'h'	işaretli kısa tamsayı (signed short int)	2
'H'	işaretsiz kısa tamsayı (unsigned short int)	2
'i'	işaretli tamsayı (signed int)	2 veya 4
'I'	işaretsiz tamsayı (unsigned int)	2 veya 4
'l'	işaretli uzun tamsayı (signed long int)	4
'L'	işaretsiz uzun tamsayı (unsigned long int)	4
'f'	kayan noktalı (float)	4
'd'	kayan noktalı (float)	8
'q'	işaretli uzun uzun tamsayı (signed long long int)	8
'Q'	işaretsiz uzun uzun tamsayı (unsigned long long int)	8

Diziler - Python’da, C Dili Benzeri Veri Tipleri

```
import array
# işaretli tamsayı dizisi
tek_sayilar  = array.array('i', [1, 3, 5, 7, 9])
print(tek_sayilar)
```

```
array('i', [1, 3, 5, 7, 9])
```

Diziler - Boyut

- Sıradan bir dizi normalde tek boyutludur. Tek boyutlu diziye güzel bir örnek, vektörlerdir. Tek boyutlu bir dizi, vektör bağlamında, çok satır ve tek sütunlu bir veri şeklinde değerlendirilir. Fakat konu vektörler değilse, genellikle, dizi veri yapısı tek satır ve çok sütunlu bir veri şeklinde düşünülür. Örneğin,

```
dizi = ['a',  'b',  'c']  # 1x3 boyutlu bir dizi
```

- dizi, 1 satır ve 3 sütundan oluşuyormuş gibi ele alınır. Bu dizi, matematikte vektör olarak kullanıldığında, 3x1 boyutlu, yani 3 satır ve 1 sütun şeklinde değerlendirilir.

Diziler - Boyut

- Diziler, tek boyuttan öte; 2, 3 ve daha çok boyutlu tanımlanabilir. İki boyutlu dizilere Matris'ler güzel bir örnektir. Benzeri bir örnek olarak bir veri tablosu alınabilir. Bir veri tablosu satır ve sütunlarında veri barındıran 2 boyutlu bir yapıdır.
- Dizi 3 ve daha fazla boyutlu ise tensör (tensor) olarak adlandırılır. Günümüzde, tensörler, yapay zeka çalışma alanlarından olan, Derin Öğrenme konusunda vb. alanlarda yaygın olarak kullanılmaktadır.
- Dizinin boyutu arttıkça işlenmesi, anlaşılması ve kodlanması güçleşir.

Diziler - Boyut

- 2x3 boyutlu bir matris :

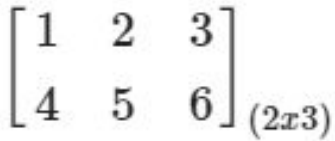
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{(2x3)}$$

```
# (2x3) boyutlu -> 2 satır 3 sütunlu matris
# 1, 2, 3
# 4, 5, 6
matris = [[1, 2, 3],
          [4, 5, 6]]
print("matris =", matris)
```

```
matris = [[1, 2, 3], [4, 5, 6]]
```

Diziler - Boyut

- Matrisin satırlarına (bazı uygulamalarda sütun) matris[n] ifadesiyle erişilir. Burada n erişilmek istenen satır (0 ile başlayarak) numarasıdır. İlk satır için matris[0], ..., vb. yazmak gerekir.



```
matris = [[1, 2, 3],
          [4, 5, 6]]
```

```
matris[0][:] # ilk satırın tüm elemanları -> [1, 2, 3]
```

- Satır ve sütun kesişimindeki konuma hücre denir. Matrisin n nci satır ve m nci sütunundaki (nxm) hücresinde matris[n-1][m-1] ifadesiyle erişilir. 2. satır ve 3. sütunundaki elemana erişelim:

```
matris[1][2] # ikinci satırın üçüncü elemanı -> 6
```

- Matrisin belirli bir sütununu liste tamamlama ifadesiyle alınabilir:

```
n = 2 # üçüncü sütun
x[n] for x in matris]  -> [3, 6]
```

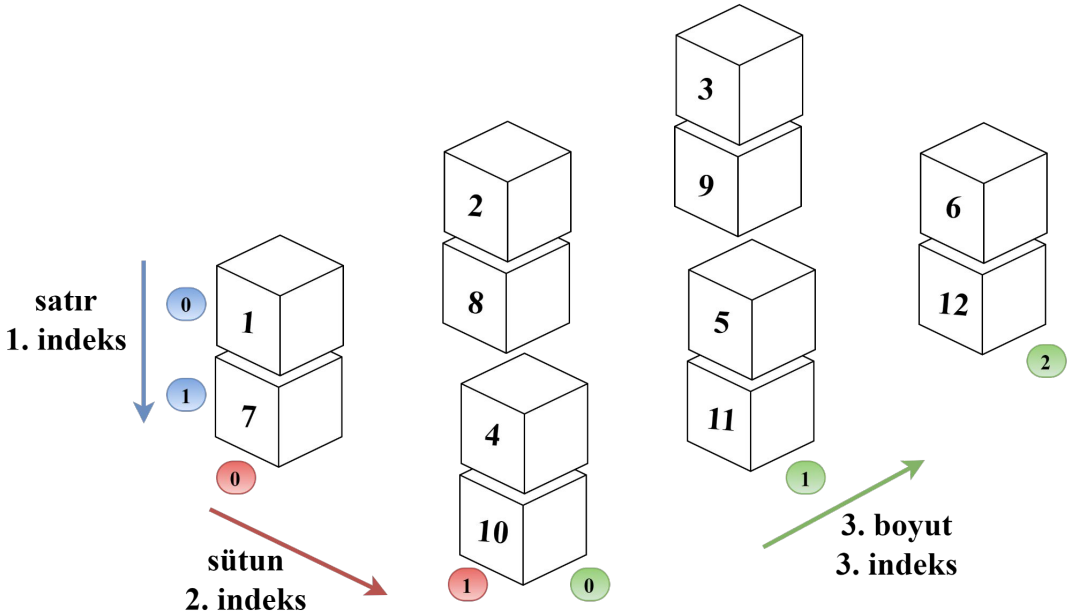
Diziler - Tensörler ve Numpy

- Vektör, matris, tensör vb. veri yapıları için, numpy kütüphanesi biçilmiş kaftandır.

  Hem kolay, hem başarımı yüksek. 2x2x3 boyutlu bir tensör örneği:

```
tensor = np.array([[[1, 2, 3], [4, 5, 6]], [[7,8,9], [10,11,12]]])
tensor.shape
(2, 2, 3)

>>>tensor
array( [ [ 1,  2,  3],
         [ 4,  5,  6],
         [ 7,  8,  9],
         [10, 11, 12] ] )
```



Diziler - Tensörler ve Numpy

- `# 2. satır, 1. sütunun tüm elemanları`

  `tensor[1, 0, :]` -> `array([7, 8, 9])`

  `tensor[1, 0, ]` -> `array([7, 8, 9])`

- `# 2. satır, 2. sütun, 3. boyuttaki 3. eleman`

  `tensor[1,1,2]` -> `12`

- `# 3. boyuttaki 2. sütun ve tüm satırlar (5 ve 11)`

  `tensor[:, 1, 1]` -> `array([ 5, 11])`

- `# 3. boyutun 2. sırasındaki`
- `# (indeks 1) tüm elemanlar`

```
tensor[:,:,1] ->
    array([[ 2,  5],
           [ 8, 11]])
```