NumPy -da İterasiya , Join , Split , Search , Sort , Filter - Part 4

NumPy -da iterasiya massiv elementlərini bir-bir nəzərdən keçirmək - ələ almaq mənasında işlədilir. NumPy-də çoxölçülü massivləri Python-da for loopundan istifadə etməklə də iterasiya edə bilərik, lakin bu, yeganə yol deyil.

1-ölçülü massivi for dövrü ilə iterasiya hər bir elementi ayrıca görəcək.

```
# 1-D dizi üzerinde iterasyon
import numpy as np

dizi = np.array([1, 2, 3, 4, 5])

for eleman in dizi:
  print(eleman)
```

```
1
2
3
4
5
```

İkiölçülü bir massivdə for dövrü bütün sətirlərin üzərindən keçəcəkdir

```
# 2-D dizi üzerinde iterasyon
import numpy as np

dizi = np.array([[1, 3, 5], [7, 9, 11]] )

for eleman in dizi:
  print(eleman)
```

```
[1 3 5]
[ 7  9 11]
```

İstənilən n -D (Dimension - Ölçülü) massiv eyni formada iterasiya olacaq. Massiv daxilindəki hər bir skalyar dəyərlərin qaytarılması üçün daxildəki massivlərin iterasiyası da tələb olunur. Məsələn, yuxarıdakı misalda 2-D massivində hər bir skalyar dəyərini qaytarmaq üçün:

```
# 2-D dizi içindeki scaler değerleri döndürmek
import numpy as np

dizi = np.array([[1, 3, 5], [7, 9, 11]] )

for eleman in dizi:
  for deger in eleman:
    print(deger)
```

```
1
3
5
7
9
11
```

3 D üçün baxaq:

```
# 3-D dizide scaler değerleri yazdırmak
import numpy as np

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

for x in dizi:
  for y in x:
    for z in y:
      print(z)
```

```
1
2
3
4
5
6
7
8
9
```

```
10
11
12
```

▾ nditer() istifadə edərək massivlərin iterasiyası

nditer() funksiyası ən sadədən ən inkişaf etmişə qədər bütün iterasiyalarda istifadə edilə bilər. Həmin funksiya iterasiyalarda qarşılaşdığımız bəzi fundamental problemləri həll edir. Xüsusilə də çoxölçülü massivlərdə, massivin hər bir skalyar vahidini təkrarlamaq n-ölçülü massivdə n ədəd for-dan istifadə etmək deməkdir ki, bu da bizdən yazılması çətin ola biləcək dövrlərdən istifadə etməyi tələb edir. Amma, nditer() istifadə edərək bu problem asanlıqla həll olunur.

```
# 3-D array de nditer() kullanımı
import numpy as np

dizi = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for deger in np.nditer(dizi):
  print(deger)
```

```
1
2
3
4
5
6
7
8
```

Massivdə hər bir elementi çap edərkən üzərindən keçərək getmək istəyiriksə, məsələn, yuxarıdakı nümunədə biz yalnız tək ədədlər əldə etmək istəyirik:

```
# 3-D array de nditer() kullanımı
# 1 değer atlayarak
import numpy as np

dizi = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for deger in np.nditer(dizi[:, :, ::2]):
  print(deger)
```

```
1
3
5
7
```

▾ ndenumerate() istifadə edərək iterasiya

enumerate, bir şeyin ardıcıl nömrəsini qeyd etmək deməkdir. Bəzən iterasiya zamanı elementin indeksinə ehtiyacımız olur, bunun üçün ndenumerate() metodundan istifadə etmək olar.

```
# ndenumerate() kullanımı
import numpy as np

dizi1D = np.array([1, 2, 3]) # 1-D dizi
dizi2D = np.array([[4, 5], [6, 7]]) # 2-D dizi

for idx, x in np.ndenumerate(dizi1D):
  print(idx, x)

for idy, y in np.ndenumerate(dizi2D):
  print(idy, y)
```

```
(0,) 1
(1,) 2
(2,) 3
(0, 0) 4
(0, 1) 5
(1, 0) 6
(1, 1) 7
```

▾ NumPy Massivlərinə Join

Join iki və ya daha çox massivin elementlərini bir massivdə birləşdirmək deməkdir. SQL-də cədvəlləri key ilə, NumPy-də massivləri axis -lərlə birləşdiririk.

Axis üzrə birləşdirmək istədiyimiz iki massivi concatenate() funksiyasına parametr olaraq veririk. Əgər Axis aşkar şəkildə təyin edilməyibsə, defolt olaraq 0-dır.

Qeyd: axis oxford lüğətinə əsasən texnoloji sahədə istiqəmtləri ifadə edir. Hansı ki, biz burada Horizontal və Vertikal (Üfüqi və Şaquli) istiqamət və ya oxları bildirir.

Istiqamət	Cədvəl üzrə	Dəyər
Şaquli - Vektorial	Sütun üzrə	axis = 0
Üfüqi - Horizontal	Sətir üzrə	axis = 1

```
# join işlemi 1-D
import numpy as np

dizi1 = np.array([1, 2, 3]) # 1-D array

dizi2 = np.array([4, 5, 6]) # 1-D array

dizi = np.concatenate((dizi1, dizi2)) # 1-D array

print(dizi)
```

```
[1 2 3 4 5 6]
```

2-D massivlər üçün eyni prosesi aparaq:

```
# join işlemi 2-D
import numpy as np

dizi1 = np.array([[1, 2], [3, 4]]) #2-D dizi

dizi2 = np.array([[5, 6], [7, 8]]) #2-D dizi

dizi = np.concatenate((dizi1, dizi2)) #2-D dizi

print(dizi)
```

```
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

Yukarıdaki örnekte 2-D diziler sütun boyunca birleştirildi. Çünkü axis değeri varsayılan olarak 0 alındı. Şimdi bu iki tane 2-D diziyi satır boyunca (axis=1) birleştirelim:

Yuxarıdakı nümunədə 2-D massivlər sütun boyunca birləşdirildi. Çünki axis dəyəri standart olaraq 0-a təyin edilmişdir. İndi gəlin bu iki ədəd 2 ölçülü massivi sətir üzrə (boyunca) birləşdirək (axis=1):

```
# join işlemi 2-D axis=1
import numpy as np

dizi1 = np.array([[1, 2], [3, 4]]) #2-D dizi

dizi2 = np.array([[5, 6], [7, 8]]) #2-D dizi

dizi = np.concatenate((dizi1, dizi2), axis=1) #2-D dizi

print(dizi)
```

```
[[1 2 5 6]
 [3 4 7 8]]
```

- Stack funksiyalarından istifadə edərək massivlərin birləşdirilməsi

  Stack mahiyyətcə concatenate ilə eynidir. Fərq ondadır ki, stack əməliyyatında birləşmə yeni ox boyunca aparılır. Məsələn, biz iki 1-D massivi ikinci ox boyunca birləşdirə bilərik ki, bu da onları bir-birinin üstünə yerləşdirir.

  Birləşdirmək istədiyimiz iki massivi axis ilə stack() funksiyasına ötürürük. Əgər ox açıq şəkildə göstərilməyibsə, standart 0-dır.

```
#stack 1-D array'lerde kullanım
import numpy as np

dizi1 = np.array([1, 2, 3])
dizi2 = np.array([4, 5, 6])

dizi_3 = np.stack((dizi1, dizi2), axis=1)
dizi_4 = np.stack((dizi1, dizi2), axis=0)
```

```
print(dizi_3) #axis=1
print(dizi_4) #axis=0
```

```
[[1 4]
 [2 5]
 [3 6]]
[[1 2 3]
 [4 5 6]]
```

NumPy-də **hstack()** sətirlər **(rows)** arasında birləşdirir/stack edir.
NumPy-də **vstack()** sütunlar **(columns)** arasında birləşmə/stack edir.
NumPy-də **dstack()** dərinlik **(depth)** boyunca birləşmə/stack edir.

```
#hstack, vstack, dstack ve 1-D array'ler
import numpy as np

dizi1 = np.array([1, 2, 3])
dizi2 = np.array([4, 5, 6])

dizi_3 = np.hstack((dizi1, dizi2))
dizi_4 = np.vstack((dizi1, dizi2))
dizi_5 = np.dstack((dizi1, dizi2))

print(dizi_3) # satırlar boyunca
print(dizi_4) # sütunlar boyunca
print(dizi_5) # derinlik/yükseklik boyunca
```

```
[1 2 3 4 5 6]
[[1 2 3]
 [4 5 6]]
[[[1 4]
  [2 5]
  [3 6]]]
```

NumPy massivlərinin bölünməsi

Split , Join -in əksidir. Join birdən çox massivi birləşdirir, Split isə massivi birdən çox massivə ayırır.
Massivləri bölmək üçün array_split() funksiyasından istifadə edirik, bu prosesdə bölmək istədiyimiz massivi və neçə hissəyə bölünəcəyini təyin edirik.

```
# splitting arrays 1-D
import numpy as np

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

yeni_dizi = np.array_split(dizi, 4) # 4 parçaya bölelim

print(yeni_dizi)
```

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

Yuxarıdakı misalda massivin bölünəcək elementlərinin sayı 12 idi və bu ədəd 4-ə bölünürdü. Bəs bu massivi 5-ə bölmək istəsək necə olar? Baxaq:

```
# splitting arrays 1-D
import numpy as np

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

yeni_dizi = np.array_split(dizi, 5) # 5 parçaya bölelim

print(yeni_dizi)
```

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

Gördüyünüz kimi, massivi bölmək üçün lazım olandan daha az elementi olduqda, bütün elementlər buna uyğun formada tənzimlənəcək.

Əslində NumPy-də split() funksiyası var, lakin verilən massivində parçalanma üçün daha az element olduqda bu funksiya, elementləri yuxarıdakı kimi təşkil etməyəcəkdir. Belə hallarda array_split() funksiyasını işlədmək daha məqsədəuyğundur. Çünki, split() funksiyası düzgün işləməyəcək - xəta alacağıq.

```
import numpy as np

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

yeni_dizi = np.split(dizi, 5) # 5 parçaya bölelim
```

```
print(yeni_dizi) # 12/5 tam bölünmediğinden split() hata verir
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-c2c6df60224b> in <module>
      3 dizi = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
      4
----> 5 yeni_dizi = np.split(dizi, 5) # 5 parçaya bölelim
      6
      7 print(yeni_dizi) # 12/5 tam bölünmediğinden split() hata verir

<__array_function__ internals> in split(*args, **kwargs)

/usr/local/lib/python3.8/dist-packages/numpy/lib/shape_base.py in split(ary, indices_or_sections, axis)
    870         N = ary.shape[axis]
    871         if N % sections:
--> 872             raise ValueError(
    873                 'array split does not result in an equal division') from None
    874     return array_split(ary, indices_or_sections, axis)

ValueError: array split does not result in an equal division
```

SEARCH STACK OVERFLOW

Array_split() böldüyü hər bir massiv elementini massiv kimi qaytarır. Massiv elementlərini aşağıdakı kimi çap edə bilərik.

```
# splitting arrays 1-D
# dizi elemanlarını yazdırma
import numpy as np

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

yeni_dizi = np.array_split(dizi, 4) # 4 parçaya bölelim

print(yeni_dizi[0])
print(yeni_dizi[1])
print(yeni_dizi[2])
print(yeni_dizi[3])
```

```
[1 2 3]
[4 5 6]
[7 8 9]
[10 11 12]
```

2-ölçülü massivdə bölünmə əməliyyatını nəzərdən keçirək. Zəhmət olmasa, aşağıdakı iki nümunəni nəzərdən keçirək və aralarındakı fərqə baxaq.

```
# splitting arrays 2-D
# dizi elemanlarını yazdırma
import numpy as np

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

yeni_dizi = np.array_split(dizi, 4) # 4 parçaya bölelim

print(yeni_dizi[0])
print(yeni_dizi[1])
print(yeni_dizi[2])
print(yeni_dizi[3])
```

```
[[1 2]
 [3 4]]
[[5 6]
 [7 8]]
[[ 9 10]]
[[11 12]]
```

```
# splitting arrays 2-D
# dizi elemanlarını yazdırma
import numpy as np

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

yeni_dizi = np.array_split(dizi, 3) # 3 parçaya bölelim

print(yeni_dizi[0])
print(yeni_dizi[1])
print(yeni_dizi[2])
```

```
[[1 2]
 [3 4]]
[[5 6]
 [7 8]]
[[ 9 10]
 [11 12]]
```

Yuxarıdakı ilk nümunə 4 ədəd 2-D massivi qaytardı. Bir elementin hər biri son iki hissəyə düşdü.

İkinci misala baxaq, bu dəfə 2-D massivlərdəki hər parça 2 elementdən ibarətdir.

Indi isə, gəlin 2-D massivin daxilindəki hər bir massivin elementlərinin sayını artıraq və nəticəni araşdıraq.

```
# splitting arrays 2-D
import numpy as np

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

yeni_dizi = np.array_split(dizi, 4) # 4 parçaya bölelim

print(yeni_dizi[0])
print(yeni_dizi[1])
print(yeni_dizi[2])
print(yeni_dizi[3])
```

```
[[1 2 3]]
[[4 5 6]]
[[7 8 9]]
[[10 11 12]]
```

```
# splitting arrays 2-D
import numpy as np

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

yeni_dizi = np.array_split(dizi, 3) # 3 parçaya bölelim

print(yeni_dizi[0])
print(yeni_dizi[1])
print(yeni_dizi[2])
```

```
[[1 2 3]
 [4 5 6]]
[[7 8 9]]
[[10 11 12]]
```

Diqqət yetirin ki, bölmə əməliyyatlarında aşağıdakı elementlər növbəti massivlərdə bərabər şəkildə yerləşdirilir!

Bölmə prosesində bölmək istədiyimiz axisi də təyin edə bilərik. Aşağıdakı nümunə 3 ədəd 2-D massivi qaytarır, lakin onlar sətir boyunca bölünür (axis = 1).

```
# splitting arrays 2-D
import numpy as np

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

yeni_dizi = np.array_split(dizi, 3, axis=1) # satır boyunca 3 parçaya bölelim

print(yeni_dizi[0])
print(yeni_dizi[1])
print(yeni_dizi[2])
```

```
[[ 1]
 [ 4]
 [ 7]
 [10]]
[[ 2]
 [ 5]
 [ 8]
 [11]]
[[ 3]
 [ 6]
 [ 9]
 [12]]
```

Yuxarıdakı misalda, sətr üzrə bölməyə alternativ olaraq, hstack() funksiyasının tərsi hsplit() ilə eyni əməli icra edir.

Eynilə, vsplit() funksiyası vstack() funksiyasının tərsi olaraq sütun wzr bölmək üçün istifadə edilə bilər.

Həmçinin, dsplit() dərinlik üzrə bölmək üçün dstack() funksiyasının əksi kimi istifadə edilə bilər. Sadəcə unutmayın ki, dsplit() funksiyası 3-D və ya daha çox ölçülü massivlər üçün istifadə olunur.

```
# splitting arrays 2-D
import numpy as np

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

yeni_dizi = np.hsplit(dizi, 3) # satır boyunca 3 parça, hsplit()

print(yeni_dizi[0])
print(yeni_dizi[1])
print(yeni_dizi[2])
```

```
[[ 1]
 [ 4]
 [ 7]
 [10]]
[[ 2]
 [ 5]
 [ 8]
 [11]]
[[ 3]
 [ 6]
 [ 9]
 [12]]
```

```
# splitting arrays 2-D
import numpy as np

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

yeni_dizi = np.vsplit(dizi, 2) # sütun boyunca 2 parça, vsplit()

print(yeni_dizi[0])
print(yeni_dizi[1])
```

```
[[1 2 3]
 [4 5 6]]
[[ 7  8  9]
 [10 11 12]]
```

- Massivlərdə axtarış

  Siz müəyyən massivdə ixtiyari dəyəri axtara və uyğun indeksləri əldə edə bilərsiniz. Massivdə dəyərləri axtarmaq üçün where() funksiyasından istifadə edirik.

```
# dizilerde search -- arama
import numpy as np

dizi = np.array([1, 2, 10, 4, 15, 6, 10, 8, 10, 1, 15])

nerede = np.where(dizi == 10) # 10 hangi indekslerde

print(nerede)
```

```
(array([2, 6, 8]),)
```

  Yuxarıdakı misalda, 10 dəyərinin olduğu elementlərin indeks nömrələri (array([2,6,8],)) tuple kimi qaytarılır. Bu o deməkdir ki, 10 dəyəri bu massivin 2-ci, 6-cı və 8-ci indekslərindədir.

  Nəzərə yetirsək görərik ki, where() funkisyası daxilində bir şərt vermişik. Bu o deməkdir ki, bu funksiya daxilində şərt strukturları istifadə edilə bilər. Məsələn, where(dizi%2==0) yazsaq, massivdəki cüt ədədlərin indekslərini verir. Eləcə də, where(dizi>=5) yazsaq, bu, bizə massivdəki 5 və 5-dən böyük rəqəmlərin indeksini verəcəkdir.

```
# dizilerde search -- arama
import numpy as np

dizi = np.array([1, 2, 10, 4, 15, 6, 10, 8, 10, 1, 15])

nerede = np.where(dizi%2 == 0) # çift sayılar hangi indekslerde

print(nerede)
```

```
(array([1, 2, 3, 5, 6, 7, 8]),)
```

- Searchsorted()

  Massivlərdə ikili axtarış edən və axtarış zamanı massivdəki ardıcıllığı qorumaq üçün göstərilən dəyərin daxil ediləcəyi indeksi bizə göstərən searchsorted() adlı funksiya var. Bu funksiya yalnız sıralanmış massivlərdə istifadə olunur.

Məsələn, aşağıdakı nümunədə massivdəki ardıcıllığı qorumaq üçün üçün 6 ədədi 5-ci indeksə əlavə edilməlidir.

Searchsorted() axtarışa soldan başlayır və 6 rəqəminin növbəti dəyərdən böyük olmadığı ilk indeksi qaytarır.

```
# dizilerde search -- arama
import numpy as np

dizi = np.array([1, 2, 3, 4, 5, 7, 8, 9, 10])

nereye = np.searchsorted(dizi, 6) # 6 nereye eklenmeli

print(nereye)
```

```
5
```

Yuxarıdakı massivin məsələn 6 ədəd elementi olsaydı searchsorted() nə qaytarardı? Baxaq:

```
# dizilerde search -- arama
import numpy as np

dizi = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) # 6 var

nereye = np.searchsorted(dizi, 6) # 6 nereye eklenmeli

print(nereye)
```

```
5
```

Yenə eyni indeksə yerləşdirildiyini gördük.
Göstərilən dəyərlərə malik bir massiv Searchsorted() vasitəsilə çoxlu dəyərləri axtarmaq üçün istifadə edilə bilər.

```
# dizilerde search -- arama
import numpy as np

dizi = np.array([1, 2, 5, 7, 9])

nereye = np.searchsorted(dizi, [3, 4, 6, 8, 10])

print(nereye)
```

```
[2 2 3 4 5]
```

Yuxarıdakı misalda qaytarılan dəyər [2, 2, 3, 4, 5]-dir. Qaytarılan dəyərlər orijinal [3, 4, 6, 8, 10] massivində, dəyərlərin yerləşdiriləcəyi indeks nömrələridir. Verilən indekslərə göndərdiyimiz massivi uyğun olaraq yerləşdirsək, massivdəki ardıcıllığ düzgün şəkildə tamamlanacaqdır.

Massivlərdə sıralama

Sorting (Sıralama) massiv elementlərinin ardıcıllığ ilə yerləşdirilməsi deməkdir. Massivdə sıralanma ədədi və ya əlifba sırası ilə, artan və ya azalan kimi üsullarla həyata keçirilir. NumPy ndarray obyektində verilmiş massivi sıralayan sort() funksiyası var. Yalnız, bu funksiya, massivin surətini yaradır, o, orijinal massivi dəyişmir.

```
# dizilerde sorting -- sıralama
import numpy as np

dizi = np.array([10, 15, 6, 4, 3, 12, 0, 1])

print(np.sort(dizi))
```

```
[ 0  1  3  4  6 10 12 15]
```

Orijinal massiv 2-D massivdirsə, sıralama hər bir alt massiv daxilində aparılır.

```
# dizilerde sorting -- sıralama
import numpy as np

dizi = np.array([[10, 15, 6, 4], [3, 12, 0, 1]]) # 2-D

print(np.sort(dizi))
```

```
[[ 4  6 10 15]
 [ 0  1  3 12]]
```

Massivlərdə Filtrləmə

Filtrləmə müəyyən bir massivdən müəyyən elementlərin götürülməsi və onlardan yeni bir massivin yaradılması deməkdir. NumPy-də filtrləmə boolean indeksi siyahısı yaratmaqla həyata keçirilə bilər.Belə ki, Boolean indeks siyahısı massivdəki indekslərə uyğun gələn məntiqi dəyərlərdir. İndeks dəyəri True olarsa, element filtrlənmiş (yəni, yeni) massivə daxil edilir, indeks dəyəri False olarsa, element filtrlənmiş massivə daxil edilmir.

```
# dizilerde filtering -- filtreleme
import numpy as np

dizi = np.array([10, 15, 6, 4, 3, 12, 0, 1]) #10,15,12 yi alalım

i=[True, True, False, False, False, True, False, False]

print(dizi[i])
```

```
[10 15 12]
```

Diqqət! true və false kimi kiçik hərflərlə True və False yazmayın.
where() funksiyasında olduğu kimi şərtdən asılı olaraq da filtrləmə edilə bilər. Aşağıdakı nümunəni nəzərdən keçirək:

```
# dizilerde filtering -- filtreleme
import numpy as np

dizi = np.array([10, 15, 6, 4, 3, 12, 0, 1])

# Boş bir dizi oluşturalım
i = []

# orijinal dizideki her bir elemana bakalım
for deger in dizi:
  if deger >= 6: #deger 6 ve 6'dan büyükse True döndürür
    i.append(True) #filtreye ekleyelim
  else:
    i.append(False) #değilse False döndürür ve eklenmez

yeni_dizi = dizi[i]

print(np.sort(yeni_dizi))
```

```
[ 6 10 12 15]
```

Yuxarıdakı nümunəyə alternativ olaraq NumPy-nin gözəlliyi ondan ibarətdir ki, biz şərti birbaşa təyin də edə bilərik. Aşağıdakı nümunəni nəzərdən keçirək:

```
#  dizilerde filtering -- filtreleme
import numpy as np

dizi = np.array([10, 15, 6, 4, 3, 12, 0, 1])

# koşulu bu şekilde de belirtebiliriz
i = (dizi%2 == 1) # tek sayıları alalım

yeni_dizi = dizi[i]

print(yeni_dizi)
```

```
[15  3  1]
```

Resurslar:
https://numpy.org/,
https://www.w3schools.com/python ,
NumPy'da İterasiya , Join , Split , Search , Sort , Filter – Bölüm 9

Bu bölümün Github resursları keçid edin.