10:09am

Numpy Practice Exercise # 2

Setup

A= np.random.randint(10,200, size=(6,12))

Part 1: Precision Indexing

1. Extract the 2nd, 4th & 6th rows only (in that order)

A[[1,3,5]]

2. Extract columns 0, 3, 7 and 11

A[np.arange(6)[:,None], np.array([0,3,7,11])]

↓ 6x1 ↓ 1x4 => 6x4

3. From the whole matrix, extract a submatrix containing:

- rows 1 to 4
- every 2nd column

α

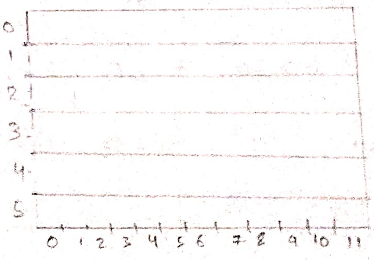
A[np.array([0,1,2,3])[:,None], ~~2~~ 1]

↓ 4x1 ↓ 1x1

4. Get the bottom-right 3x4 block

6-3 , 12-4

3 8



A[np.arange(3,6)[:,None],
np.arange(8,12)]

3x1 1x4

= 3x4

Part 2: Smart Slicing

5. Reverse only the middle 4 rows

A[1:5] = A[4:0:-1]

A

6. Keep the first half of colums, discard the rest

A[:, : ~~7~~ 6]

7. Swap the first & last rows

```
A[0::5] = A[5::-5]
```

8. Swap the first & last columns

```
A[:, 0::11] = A[:, -1::-12]
```

-> looks good to me

just trying -ve indexing

-1-1=-2
-1-12=-13 - does not exist in A

-1 X -12

Part 3: Boolean & Mask logic

9. Creat a mask for values divisible by 5

```
A%5==0
```

10. Set all values greater than 150 to 999

```
A[A>150] = 999
```

11. Extract elements that are
- greater than 50
- AND less than 120

```
A[(A>50) & (A<120)]
```

12. Count how many elements are even

```
A[A%2==0].shape[0]
```

Part 4: Sorting with Control

13. Sort only the last 3 columns of the array

```
A[:, -1:-4:-1] = np.sort(A[:, 9:], axis=0)
```

14. Sort rows based on their maximum value

```
a= np.sum(A, axis=1)
b= np.argsort(a)
A[b]
```

15. Sort
a=
b=
A[b

16. For

Part 5:
14 with
• For
elem
a=np.

(6,)

18. Reor
the
a=np
A[

19. Re
res
a

15. Sort rows based on their mean value

```
a = np.mean(A, axis=1)
b = np.argsort(a)
A[b]
```

16. For each column find the index of its smallest value

```
np.argsort(A, axis=0)[:,0]
```

→ Applies [:,0] on a 6×12 returns ~~6×6~~ (6,) array

This array contains the smallest value of only 6th column values! There are 12 columns

```
np.argsort(A, axis=0)[0]
```

Part 5: Thinking section

17 without using loops

- For each row, subtract the row mean from every element

```
a = np.mean(A, axis=1)
```

A - a

~~6×12~~ 12 (6,)

or

a - A

(6,) ~~6×12~~ 12

or

```
np.subtract(A, a)
```

→ Dunno if this will work my first time doin this

Would not broadcast

```
a = np.mean(A, axis=1)[:, None]
```

↳ 6×1

if ∄ a subtract func

⇒ A - a

18. Reorder columns based on the sorted order of the last row.

```
a = np.argsort(A[-1])
A[:, a]
```

19. Keep only the largest 2 values per column; set the rest to -1

```
a = np.argsort(A, axis=0) ~~[:, 4:]~~ [4:].T
```

↳ ~~12×6~~ 6×12 [:, 4:] ← 12×2

```
b = np.zeros_like(A)
```

~~b[:, 4:]~~

~~b[np.arange(12)[:, None], a] = 1~~

```
b[a, np.arange(12)[:, None]] = 1
```

12×1 12×2

6×12 12×2 12×6

~~B[:,1]~~

mask = b < 1

A[mask] = 0

20. Create a new matrix where:
- rows with sum > overall mean sum are kept
- others are removed

mask1 = np.sum(A, axis=1) > np.mean(A)

assuming mean is calculated row-wise at default

A[mask1]

worked because

mask1 is 1D & is applied on axis 0 (Row)

mask1	A
(6,)	(6,12)

matched

If you want to match columns than apply mask1 on axis 1 (colum)

A[:, mask1]

& col(A) = 6 × 12

must match with length of mask1' colsize?

Setup

B = np.ra

Part 1: Cont

1. Extract
- every
- every

B[0::2, 0

2. Extract
tive

a = B[:,
B[a<0

3. From
center

4. E