Introduction to computing and data handling 2023

Back to numpy array

Numpy arrays are powerful tools for numerical computing*. So far we have only seen 1-dimensional arrays. It is possible to create arrays of any dimensions.

Let us create

>>>y = np.array([[2., 15., -5., 0.9])

>>>z = np.array([[2., 15., -5., 0.9],[10., 25., 8., 0.001]])

Use functions

len

np.shape

np.size

and see how the results differ. Which function will you use to find out the dimensions of an n-dimension array ?

Create an n dimension array of arbitrary n using *np.reshape*

>>>y.reshape(2,2)

>>>w = np.arange(1,5)

>>>w.reshape(2,2)

Follow the same method, use arange and create a 3 dimensional array.

---

* Numpy arrays can be of strings, boolean, and complex data types. But we'll focus on floats and integers.

Array slicing.

To slice a numpy array, the syntax is *a[start:end:step, start:end:step, …..]*
The slicing details on each axis is given separated by comma. The start:end:step will give the details of how an axis should be sliced. That is, *[start:end:step]* will result in index = *start, start+step, start+2.*step…. start+n*step,* where *start+n*step< end*

Let us take it as a write it in this form. The below is a 3X4 matrix.

a_{00}	a_{01}	a_{02}	a_{03}	**Zeroth row**
a_{10}	a_{11}	a_{12}	a_{13}	**First row**
a_{20}	a_{21}	a_{22}	a_{23}	**Second row**
Column 0	**Column 1**	**Column 2**	**Column 3**	

Let us say, row number is decided by index j and column number by index k.
Operation [0:1:1] on row will give row number 0, ie, j=0, operation [0:1:1] on column will give column number 0, ie., k=0 leading to a_{00} as the answer. If you consider [0:3:2] on the row, the row indices resulted are j=0,2. If you consider [1:3:1] on the column, it will lead to k=1,2 (3 is not included that is, the end index is not included)†. Therefore the final result will be

a_{01} a_{02}

a_{21} a_{22}

Example: Use the numpy array *slice_test* given below and make various slices.
This matrix has three rows and 4 columns
>>>*np.shape(slice_test)* #will give you(3,4)

---

† You can get these by doing *np.arange*. For example, *np.arange(0,3,2)* = 0,2. *np.arange(1,3,1)* = 1,2

2.0	15.0	-5.0	0.9	Zeroth row
10	25	8	0.001	First row
12.0	89	0.5	-2.4	Second row
Column 0	Column 1	Column 2	Column 3	

Try out different slices and see what you get. On the other way round, see what operation you need to get a particular slice.

For example,

(i) The above slicing of will lead to a 2X2 matrix

15.0 -5.0

89. 0.5

(ii) z [0:1:1,0:1:1] will lead to the zeroth row and zeroth column, i.e, just element a_{00}=2.

(iii) See what slicing operation will result in

-5.0

8.0

(iv) If you do not specify, the default values of start(0), end(last), and stop(1) will be used.

(v) What does z[0][0] give? Why?

(vi) A step of -1 reverses the order.

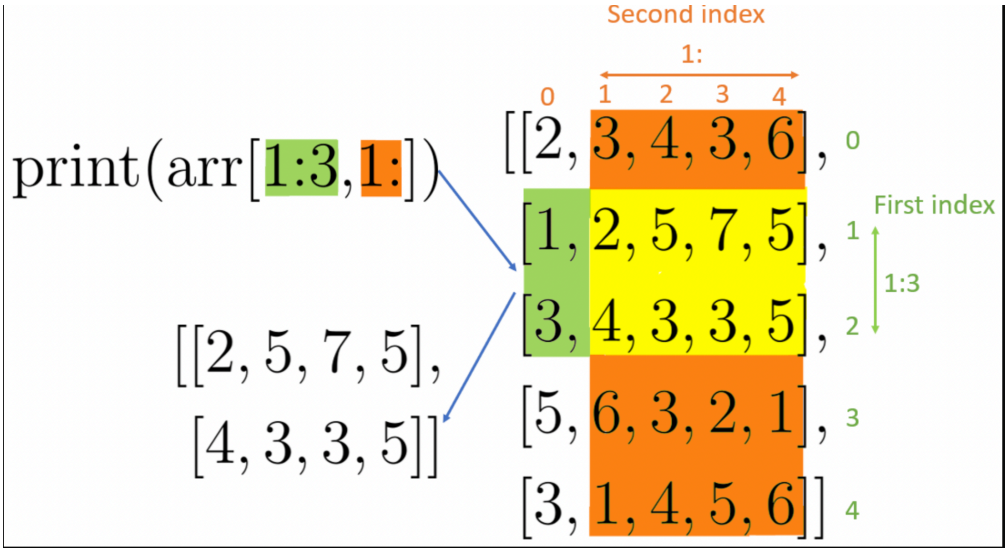
Find the dot product of *a1 = [[1,2],[3,4]]* and *a2 = [[5,6],[7,8]]* guided by the indices of the elements. Verify your result using *numpy.dot* function.

A simple approach is,

- define a1[0] and a2[:,0] as two arrays. Both will have two elements.
- run a for loop of two steps where the first elements and the second elements are multiplied (no cross multiplication), leading to two numbers.
- Find the sum of the two numbers.
- See if this can be done more efficiently than the method above.

4th class *numpy*, plotting continuing. 4th Sep

Some visualizations to understand how the indexing & slicing works.




```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```