|  |  |
| --- | --- |
| **Installing, loading and checking version of numpy**  **NUMPY CHEAT SHEET**  >>> conda install numpy  >>> !pip install numpy  >>> import numpy as np  >>> np.version.version  >>> np.\_\_version\_\_ | **OUTPUTS** |
| **Creating 1D, 2D and 3D arrays**  >>> arr1 = np.array((1,2,3,4,5,6))  >>> arr2 = np.array([1,2,3,4,5,6])  >>> array1D = np.array (range(50,100))  >>> array2D = np.arange(50,100).reshape(2,25)  >>> array3D = np.arange(50,100).reshape(5,2,5)  >>> arr1d = np.array([1,2,3,4,5,6])  >>> arr2d = np.array([[1,2,3],[4,5,6]])  >>> arr3d = np.array([[[1,2],[3]],[[4,5],[6]]])  >>> array1 **=** np**.**array([[1,2,3],[4,5,6],[7,8,9]])  >>> array2 **=** np**.**array([[11,12,13],[14,15,16],[17,18,19]]) | 1. array([1, 2, 3, 4, 5, 6]) 2. array([1, 2, 3, 4, 5, 6]) 3. array([50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83,84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]) 4. array([[50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,66, 67, 68, 69, 70, 71, 72, 73, 74],   [75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,  91, 92, 93, 94, 95, 96, 97, 98, 99]])   1. array([[[50, 51, 52, 53, 54],   [55, 56, 57, 58, 59]],  [[60, 61, 62, 63, 64],  [65, 66, 67, 68, 69]],  [[70, 71, 72, 73, 74],  [75, 76, 77, 78, 79]],  [[80, 81, 82, 83, 84],  [85, 86, 87, 88, 89]],  [[90, 91, 92, 93, 94],  [95, 96, 97, 98, 99]]]) |
| **Built-In FUNCTIONS**  1. dtype -----> type of the data used inside the numpy arrays  >>> array1D.dtype  >>> array1D = np.array (range(50,100), dtype = 'str')  2. shape -----> structure of the numpy arrays  >>> array1D.shape  3. range -----> in between the start and stop values  >>> array1D = np.array (range(50,100))  4. ndim -----> tells whether it is 1D, 2D, 3D or so on  >>> array1D.ndim  5. ndmin -----> instruct you to take those dimensions inside the numpy array  >>> array1D = np.array (range(50,100) , ndmin = 25)  6. size -----> total number of elements  >>> array1D.size  7. nbytes -----> total number of bits (1 byte = 4 bits)  >>> array1D.nbytes  8. reshape -----> reshaping the numpy array into requried dimesntions  >>> array1D = np.array (range(50,100)).reshape (5,5,2) | 1. dtype('int32') 2. (50,) 3. array([50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83,84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]) 4. 1 5. 25 6. 50 7. 200 8. array([[[50, 51, 52, 53, 54],   [55, 56, 57, 58, 59]],  [[60, 61, 62, 63, 64],  [65, 66, 67, 68, 69]],  [[70, 71, 72, 73, 74],  [75, 76, 77, 78, 79]],  [[80, 81, 82, 83, 84],  [85, 86, 87, 88, 89]],  [[90, 91, 92, 93, 94],  [95, 96, 97, 98, 99]]]) |
| **SPECIAL FUNCTIONS**  1. arange -----> in between the start and stop(excluded) with specified float vlaue steps  >>> array2D = np.arange(50,100,5.2).reshape(2,5)  2. linespace -----> in between the start and stop(included) with specified float vlaue steps  >>> array2 = np.linspace(50,100,10).reshape((2,5))  3. zero -----> np.zeros((r,c)) array with the specified dimensions and data is filled with zeros.  >>> array1D = np.zeros((1,))  4. ones -----> np.ones((r,c)) array with the specified dimensions and data is filled with ones.  >>> array1D = np.ones((1,))  5. full -----> np.full((r,c),n) array with the specified dimensions and data is filled with n(num/str).  >>> array1D = np.full((1,),'1d')  6. eye -----> array where all elements are equal to zero, except for the k-th diagonal, whose values are equal to one. This creates the identity array.  array2D = np.eye(10 , dtype = 'int32')  7. identity -----> generates square array with ones on the main diagonal  >>> array2D = np.identity(10 , dtype = 'int32')  8. diag() -----> function extract or construct diagonal array.  >>> array2D = np.diag (np.arange(5,20))  9. flipud -----> flipping in up-down direction  >>> np.flipud(array2D)  10. fliplr -----> flipping in left-right direction  >>> np.fliplr(array2D)  11. flip -----> flipping in reverse direction  >>> np.flip(array2D)  12. flip -----> flipping in axis=1 direction  >>> np.flip(array2D , axis = 1)  13. flip -----> flipping in axis=0 direction  >>> np.flip(array2D , axis = 0)  14. transpose -----> changing rows into columns and vice-versa  >>> array1D.T  >>> np.transpose(array1D)  15. rot90 -----> changing rows into columns in anti-clockwise direction of 90's  >>> np.rot90(array3D)  >>> np.flipud(np.transpose(array3D)) | 1. array([50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83,84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]) 2. array([[ 50. , 55.55555556, 61.11111111, 66.66666667,72.22222222],[ 77.77777778, 83.33333333, 88.88888889, 94.44444444,100.]]) 3. array([0.]) 4. array([1.]) 5. array(['1d'], dtype='<U2') 6. array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],   [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],  [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],  [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],  [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],  [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],  [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],  [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])   1. array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],   [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],  [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],  [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],  [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],  [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],  [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],  [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])   1. array([[ 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],   [ 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  [ 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  [ 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  [ 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  [ 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0],  [ 0, 0, 0, 0, 0, 0, 11, 0, 0, 0, 0, 0, 0, 0, 0],  [ 0, 0, 0, 0, 0, 0, 0, 12, 0, 0, 0, 0, 0, 0, 0],  [ 0, 0, 0, 0, 0, 0, 0, 0, 13, 0, 0, 0, 0, 0, 0],  [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 14, 0, 0, 0, 0, 0],  [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 0, 0, 0, 0],  [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 16, 0, 0, 0],  [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 0],  [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 18, 0],  [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 19]])   1. array([[74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59,58, 57, 56, 55, 54, 53, 52, 51, 50],   [99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84,83, 82, 81, 80, 79, 78, 77, 76, 75]])   1. array([[75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,91, 92, 93, 94, 95, 96, 97, 98, 99],   [50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,66, 67, 68, 69, 70, 71, 72, 73, 74]])   1. array([[99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84,83, 82, 81, 80, 79, 78, 77, 76, 75],   [74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59,58, 57, 56, 55, 54, 53, 52, 51, 50]])   1. array([[74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59,58, 57, 56, 55, 54, 53, 52, 51, 50],   [99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84,83, 82, 81, 80, 79, 78, 77, 76, 75]])   1. array([[75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,91, 92, 93, 94, 95, 96, 97, 98, 99],   [50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,66, 67, 68, 69, 70, 71, 72, 73, 74]])   1. array([50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83,84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]) 2. array([[[55, 56, 57, 58, 59],   [65, 66, 67, 68, 69],  [75, 76, 77, 78, 79],  [85, 86, 87, 88, 89],  [95, 96, 97, 98, 99]],  [[50, 51, 52, 53, 54],  [60, 61, 62, 63, 64],  [70, 71, 72, 73, 74],  [80, 81, 82, 83, 84],  [90, 91, 92, 93, 94]]]) |
| **INDEXING AND SLICING**   1. array1D 2. array1D[0] 3. array1D[**-**1] 4. array1D[::5] 5. array1D[::**-**10] 6. array2D 7. array2D[0:2] 8. array2D[0:2 , 1] 9. array2D[0:2][1] 10. array3D 11. array3D[0:2] 12. array3D[0:2 , 1] 13. array3D[0:2][1] | 1. array([50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83,84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]) 2. 50 3. 99 4. array([50, 55, 60, 65, 70, 75, 80, 85, 90, 95]) 5. array([99, 89, 79, 69, 59]) 6. array([[50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,64,65,66, 67, 68, 69, 70, 71, 72, 73, 74],   [75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,91, 92, 93, 94, 95, 96, 97, 98, 99]])   1. array([[50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74],   [75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,91, 92, 93, 94, 95, 96, 97, 98, 99]])   1. array([51, 76]) 2. array([75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,92, 93, 94, 95, 96, 97, 98, 99]) 3. array([[[50, 51, 52, 53, 54],   [55, 56, 57, 58, 59]],  [[60, 61, 62, 63, 64],  [65, 66, 67, 68, 69]],  [[70, 71, 72, 73, 74],  [75, 76, 77, 78, 79]],  [[80, 81, 82, 83, 84],  [85, 86, 87, 88, 89]],  [[90, 91, 92, 93, 94],  [95, 96, 97, 98, 99]]])   1. array([[[50, 51, 52, 53, 54],   [55, 56, 57, 58, 59]],  [[60, 61, 62, 63, 64],  [65, 66, 67, 68, 69]]])   1. array([[55, 56, 57, 58, 59],   [65, 66, 67, 68, 69]])   1. array([[60, 61, 62, 63, 64],   [65, 66, 67, 68, 69]]) |
| **STATISTICAL FUNCTIONS**  1. max -----> maximum in array  >>> array1D.max()  2. max -----> maximum in array via axis = 1  >>> array2D.max(axis = 1)  3. max -----> maximum in array via axis = 0  >>> array2D.max(axis = 0)  4. min -----> minimum in array  >>> array1D.min()  5. min -----> minimum in array via axis = 1  >>> array2D.min(axis = 1)  6. min -----> minimum in array via axis = 0  >>> array2D.min(axis = 0)  7. sum -----> sum of all elements  >>> array1D.sum()  8. sum -----> sum of elements in array via axis = 1  >>> array2D.sum(axis = 1)  9. sum -----> sum of elements in array via axis = 0  >>> array2D.sum(axis = 0)  10. mean -----> sum of all elements divided by size of elements  >>> array1D.mean()  11. median -----> average of the middle elements  >>> np.median(array1D)  12. varience -----> sum of (x-u)\*\*2 divided by n  >>> np.var(array1D)  13. standard deviation -----> sqrt of (sum of (x-u)\*\*2 divided by n)  >>> np.std(array1D) | 1. 99 2. array([74, 99]) 3. array([75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]) 4. 50 5. array([50, 75]) 6. array([50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74]) 7. 3725 8. array([1550, 2175]) 9. array([125, 127, 129, 131, 133, 135, 137, 139, 141, 143, 145, 147, 149,151, 153, 155, 157, 159, 161, 163, 165, 167, 169, 171, 173]) 10. 74.5 11. 74.5 12. 208.25 13. 14.430869689661812 |
| **STACKING and SPLITTING**  1. vstack -----> vertical attachment (columns are equal in 2 arrays)  >>> np.vstack((array1,array2))  2. hstack -----> horizontal attachment (rows are equal in 2 arrays)  >>> np.hstack((array1,array2))  3. dstack -----> parllel elements in 2 arrays along the third axis (min 3D array is required)  >>> np.dstack((array1,array2))  4. hsplit -----> horizontal split means dividing columns  >>> np.hsplit(array2D,5)  5. vsplit -----> vertical split means dividing rows  >>> np.vsplit(array2D,1)  6. where -----> returns values with some conditions  >>> np.where(array2D%5==0, array2D , 'False') | 1. array([[ 1, 2, 3],  [ 4, 5, 6],  [ 7, 8, 9],  [11, 12, 13],  [14, 15, 16]])  2. array([[ 1, 2, 3, 11, 12, 13],  [ 4, 5, 6, 14, 15, 16],  [ 7, 8, 9, 17, 18, 19]])  3. array([[[ 1, 11],  [ 2, 12],  [ 3, 13]]])  4. [array([[50, 51, 52, 53, 54],  [75, 76, 77, 78, 79]]),  array([[55, 56, 57, 58, 59],  [80, 81, 82, 83, 84]]),  array([[60, 61, 62, 63, 64],  [85, 86, 87, 88, 89]]),  array([[65, 66, 67, 68, 69],  [90, 91, 92, 93, 94]]),  array([[70, 71, 72, 73, 74],  [95, 96, 97, 98, 99]])]  5. [array([[50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64,  65,66, 67, 68, 69, 70, 71, 72, 73, 74],  [75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,  91, 92, 93, 94, 95, 96, 97, 98, 99]])]  6. array([['50', 'False', 'False', 'False', 'False', '55', 'False', 'False',  'False', 'False', '60', 'False', 'False', 'False', 'False', '65',  'False', 'False', 'False', 'False', '70', 'False', 'False',  'False', 'False'],  ['75', 'False', 'False', 'False', 'False', '80', 'False', 'False',  'False', 'False', '85', 'False', 'False', 'False', 'False', '90',  'False', 'False', 'False', 'False', '95', 'False', 'False',  'False', 'False']], dtype='<U11') |
| **SET OPERATIONS**  1. union1d -----> combining  >>> np.union1d(array1,array2)  2. intersect1d -----> common  >>> np.intersect1d(array1,array2)  3. setdiff1d -----> subtracting array2 from array1 and printing remaining elements in array1  >>> np.setdiff1d(array1,array2) | 1. array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18,  19])  2. array([], dtype=int32)  3. array([1, 2, 3, 4, 5, 6, 7, 8, 9]) |
| **ARITHMETIC OPERATIONS**  array1 = np.array([[1,2,3],[4,5,6],[7,8,9]])  array2 = np.array([[11,12,13],[14,15,16],[17,18,19]])   1. array1 + array2 2. array1 - array2 3. array1 \* array2 4. array1 / array2 5. array1 // array2 6. array1 % array2 7. array1 @ array2 | 1.array([[12, 14, 16],  [18, 20, 22],  [24, 26, 28]])  2.array([[-10, -10, -10],  [-10, -10, -10],  [-10, -10, -10]])  3.array([[ 11, 24, 39],  [ 56, 75, 96],  [119, 144, 171]])  4.array([[0.09090909, 0.16666667, 0.23076923],  [0.28571429, 0.33333333, 0.375 ],  [0.41176471, 0.44444444, 0.47368421]])  5.array([[0, 0, 0],  [0, 0, 0],  [0, 0, 0]])  6.array([[1, 2, 3],  [4, 5, 6],  [7, 8, 9]])  7.array([[ 90, 96, 102],  [216, 231, 246],  [342, 366, 390]]) |
| **AGGREGATE FUNCTIONS**  a1 = np.array([[1,2,3],[4,5,6],[7,8,9]])  a2 = np.array([[11,12,13],[14,15,16],[17,18,19]])   1. np.add(a1,a2) 2. np.subtract(a1,a2) 3. np.multiply(a1,a2) 4. np.matmul(a1,a2) | 1.array([[12, 14, 16],  [18, 20, 22],  [24, 26, 28]])  2.array([[-10, -10, -10],  [-10, -10, -10],  [-10, -10, -10]])  3.array([[ 11, 24, 39],  [ 56, 75, 96],  [119, 144, 171]])  4.array([[ 90, 96, 102],  [216, 231, 246],  [342, 366, 390]]) |
| **APPEND AND CONCATENATE**  1. append -----> adding at the last of the array  2. concatenate -----> adding in left-right or up-down direction  a1 = np.array([[1,2,3],[4,5,6],[7,8,9]])  a2 = np.array([[11,12,13],[14,15,16],[17,18,19]])   1. np.append(a1,a2)   np.append(a1,a2,axis = 1)  np.append(a1,a2,axis = 0)   1. np.concatenate((a1,a2))   np.concatenate((a1,a2) , axis = 1)  np.concatenate((a1,a2) , axis = 0) | 1. array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18,  19])  2. array([[ 1, 2, 3],  [ 4, 5, 6],  [ 7, 8, 9],  [11, 12, 13],  [14, 15, 16],  [17, 18, 19]]) |
| **FILTERING ARRAYS**  a1 = np.array([[1,2,3],[4,5,6],[7,8,9]])  a2 = np.array([[11,12,13],[14,15,16],[17,18,19]])   1. a1 > 5 2. a2 < 15 | 1. array([[False, False, False],   [False, False, True],  [ True, True, True]])   1. array([[ True, True, True],   [ True, False, False],  [False, False, False]]) |
| **Numpy RANDOM NUMBERS**  1. np.random.rand -----> generates an array with random numbers that are uniformly distributed between 0 and 1  >>> array2D = np.random.rand(2,3)  2. np.random.randn -----> generates an array with random numbers that are normally distributed with mean = 0 and sd = 1  >>> array2D = np.random.randn(2,3)  3. np.random.randint -----> generates an array with random numbers that are uniformly distributed between 0 and given integer  >>> array2D = np.random.randint(50 , size = (2,3))  4. np.random.uniform -----> generates array with random numbers that are uniformly distributed within the given range of values  >>> array2D = np.random.uniform(50,60 , size = (2,3))  5. np.random.seed -----> puts the random values constant even though we execute the random code for multiple times  >>> np.random.seed(1372)  >>> array2D = np.random.randint(50 , size = (2,3)) | 1. array([[0.73852266, 0.13648889, 0.53915898],  [0.52860591, 0.01443914, 0.09355033]])  2. array([[-0.72506201, 0.06380681, -1.17170102],  [-0.29041479, -0.80791029, 0.75431523]])  3. array([[22, 49, 34],  [48, 35, 36]])  4. array([[58.08107549, 59.19999339, 50.4455265 ],  [59.53420965, 57.62848807, 52.49294163]])  5. array([[20, 2, 42],  [ 7, 25, 46]]) |
| **EXPANDING AND SQUEEZING**  1. expand\_dims() -----> can add a new axis to an array using the expand\_dims() method by providing the array and the axis along which to expand  a1 = np.array([1,2,3,4,5,6,7,8,9])  a1  a1.ndim  a2 = np.expand\_dims(a1 , axis = 0)  a2  a2.ndim  a2 = np.expand\_dims(a1 , axis = 1)  a2  a2.ndim  2. squeeze() -----> removes the axis that has a single entry  a1 = np.array([[[1,2,3,4,5,6,7,8,9]]])  a1  a1.ndim  a2 = np.squeeze(a1 , axis = 0)  a2  a2.ndim  a2 = np.squeeze(a1 , axis = 1)  a2  a2.ndim | **#expanding**  array([1, 2, 3, 4, 5, 6, 7, 8, 9])  1  array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])  2  array([[1],  [2],  [3],  [4],  [5],  [6],  [7],  [8],  [9]])  2  **#squeezing**  array([[[1, 2, 3, 4, 5, 6, 7, 8, 9]]])  3  array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])  2  array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])  2 |