# NUMPY CHEAT SHEET

| **Installing, loading and checking version of numpy**<br><br>>>> conda install numpy<br>>>> !pip install numpy<br>>>> import numpy as np<br>>>> np.version.version<br>>>> np.__version__ | **OUTPUTS** |
|---|---|
| **Creating 1D, 2D and 3D arrays**<br><br>>>> arr1 = np.array((1,2,3,4,5,6))<br>>>> arr2 = np.array([1,2,3,4,5,6])<br>>>> array1D = np.array (range(50,100))<br>>>> array2D = np.arange(50,100).reshape(2,25)<br>>>> array3D = np.arange(50,100).reshape(5,2,5)<br>>>> arr1d = np.array([1,2,3,4,5,6])<br>>>> arr2d = np.array([[1,2,3],[4,5,6]])<br>>>> arr3d = np.array([[[1,2],[3]],[[4,5],[6]]])<br>>>> array1 = np.array([[1,2,3],[4,5,6],[7,8,9]])<br>>>> array2 = np.array([[11,12,13],[14,15,16],[17,18,19]]) | 1. array([1, 2, 3, 4, 5, 6])<br>2. array([1, 2, 3, 4, 5, 6])<br>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])<br>4. array([[50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 6 5,66, 67, 68, 69, 70, 71, 72, 73, 74],<br>[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]])<br>5. array([[[50, 51, 52, 53, 54],<br>[55, 56, 57, 58, 59]],<br>[[60, 61, 62, 63, 64],<br>[65, 66, 67, 68, 69]],<br>[[70, 71, 72, 73, 74],<br>[75, 76, 77, 78, 79]],<br>[[80, 81, 82, 83, 84],<br>[85, 86, 87, 88, 89]],<br>[[90, 91, 92, 93, 94],<br>[95, 96, 97, 98, 99]]]) |
| **Built-In FUNCTIONS**<br><br>1. dtype -----> type of the data used inside the numpy arrays<br>>>> array1D.dtype<br>>>> array1D = np.array (range(50,100), dtype = 'str')<br>2. shape -----> structure of the numpy arrays<br>>>> array1D.shape<br>3. range -----> in between the start and stop values<br>>>> array1D = np.array (range(50,100))<br>4. ndim -----> tells whether it is 1D, 2D, 3D or so on<br>>>> array1D.ndim<br>5. ndmin -----> instruct you to take those dimensions inside the numpy array<br>>>> array1D = np.array (range(50,100) , ndmin = 25)<br>6. size -----> total number of elements<br>>>> array1D.size<br>7. nbytes -----> total number of bits (1 byte = 4 bits)<br>>>> array1D.nbytes<br>8. reshape -----> reshaping the numpy array into requried dimesntions<br>>>> array1D = np.array (range(50,100)).reshape (5,5,2) | 1. dtype('int32')<br>2. (50,)<br>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])<br>4. 1<br>5. 25<br>6. 50<br>7. 200<br>8. array([[[50, 51, 52, 53, 54],<br>[55, 56, 57, 58, 59]],<br>[[60, 61, 62, 63, 64],<br>[65, 66, 67, 68, 69]],<br>[[70, 71, 72, 73, 74],<br>[75, 76, 77, 78, 79]],<br>[[80, 81, 82, 83, 84],<br>[85, 86, 87, 88, 89]],<br>[[90, 91, 92, 93, 94],<br>[95, 96, 97, 98, 99]]]]) |
| **SPECIAL FUNCTIONS**<br><br>1. arange -----> in between the start and stop(excluded) with specified float vlaue steps<br>>>> array2D = np.arange(50,100,5.2).reshape(2,5)<br>2. linespace -----> in between the start and stop(included) with specified float vlaue steps<br>>>> array2 = np.linspace(50,100,10).reshape((2,5))<br>3. zero -----> np.zeros((r,c)) array with the specified dimensions and data is filled with zeros.<br>>>> array1D = np.zeros((1,))<br>4. ones -----> np.ones((r,c)) array with the specified dimensions and data is filled with ones.<br>>>> array1D = np.ones((1,))<br>5. full -----> np.full((r,c),n) array with the specified dimensions and data is filled with n(num/str).<br>>>> array1D = np.full((1,),'1d') | 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])<br>2. array([[ 50.        , 55.55555556, 61.11111111, 66.66666667, 72.22222222],[ 77.77777778, 83.33333333, 88.8888888 9, 94.44444444,100.]])<br>3. array([0.])<br>4. array([1.])<br>5. array(['1d'], dtype='<U2')<br>6. array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],<br>[0, 1, 0, 0, 0, 0, 0, 0, 0, 0],<br>[0, 0, 1, 0, 0, 0, 0, 0, 0, 0],<br>[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],<br>[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],<br>[0, 0, 0, 0, 0, 1, 0, 0, 0, 0],<br>[0, 0, 0, 0, 0, 0, 1, 0, 0, 0],<br>[0, 0, 0, 0, 0, 0, 0, 1, 0, 0],<br>[0, 0, 0, 0, 0, 0, 0, 0, 1, 0], |

| | |
|---|---|
| 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.<br>array2D = np.eye(10 , dtype = 'int32')<br>7. identity -----> generates square array with ones on the main diagonal<br>>>> array2D = np.identity(10 , dtype = 'int32')<br>8. diag() -----> function extract or construct diagonal array.<br>>>> array2D = np.diag (np.arange(5,20))<br>9. flipud -----> flipping in up-down direction<br>>>> np.flipud(array2D)<br>10. fliplr -----> flipping in left-right direction<br>>>> np.fliplr(array2D)<br>11. flip -----> flipping in reverse direction<br>>>> np.flip(array2D)<br>12. flip -----> flipping in axis=1 direction<br>>>> np.flip(array2D , axis = 1)<br>13. flip -----> flipping in axis=0 direction<br>>>> np.flip(array2D , axis = 0)<br>14. transpose -----> changing rows into columns and vice-versa<br>>>> array1D.T<br>>>> np.transpose(array1D)<br>15. rot90 -----> changing rows into columns in anti-clockwise direction of 90's<br>>>> np.rot90(array3D)<br>>>> np.flipud(np.transpose(array3D)) | [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])<br>7. array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],<br>[0, 1, 0, 0, 0, 0, 0, 0, 0, 0],<br>[0, 0, 1, 0, 0, 0, 0, 0, 0, 0],<br>[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],<br>[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],<br>[0, 0, 0, 0, 0, 1, 0, 0, 0, 0],<br>[0, 0, 0, 0, 0, 0, 1, 0, 0, 0],<br>[0, 0, 0, 0, 0, 0, 0, 1, 0, 0],<br>[0, 0, 0, 0, 0, 0, 0, 0, 1, 0],<br>[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])<br>8. array([[ 5,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],<br>[ 0,  6,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],<br>[ 0,  0,  7,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],<br>[ 0,  0,  0,  8,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],<br>[ 0,  0,  0,  0,  9,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],<br>[ 0,  0,  0,  0,  0, 10,  0,  0,  0,  0,  0,  0,  0,  0,  0],<br>[ 0,  0,  0,  0,  0,  0, 11,  0,  0,  0,  0,  0,  0,  0,  0],<br>[ 0,  0,  0,  0,  0,  0,  0, 12,  0,  0,  0,  0,  0,  0,  0],<br>[ 0,  0,  0,  0,  0,  0,  0,  0, 13,  0,  0,  0,  0,  0,  0],<br>[ 0,  0,  0,  0,  0,  0,  0,  0,  0, 14,  0,  0,  0,  0,  0],<br>[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 15,  0,  0,  0,  0],<br>[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 16,  0,  0,  0],<br>[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 17,  0,  0],<br>[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 18,  0],<br>[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 19]])<br>9. array([[74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60<br>, 59,58, 57, 56, 55, 54, 53, 52, 51, 50],<br>[99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84<br>,83, 82, 81, 80, 79, 78, 77, 76, 75]])<br>10. array([[75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89<br>, 90,91, 92, 93, 94, 95, 96, 97, 98, 99],<br>[50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65<br>,66, 67, 68, 69, 70, 71, 72, 73, 74]])<br>11. array([[99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85<br>, 84,83, 82, 81, 80, 79, 78, 77, 76, 75],<br>[74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59<br>,58, 57, 56, 55, 54, 53, 52, 51, 50]])<br>12. array([[74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60<br>, 59,58, 57, 56, 55, 54, 53, 52, 51, 50],<br>[99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84<br>,83, 82, 81, 80, 79, 78, 77, 76, 75]])<br>13. array([[75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89<br>, 90,91, 92, 93, 94, 95, 96, 97, 98, 99],<br>[50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65<br>,66, 67, 68, 69, 70, 71, 72, 73, 74]])<br>14. array([50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64,<br>65, 66,67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,<br>81, 82, 83,84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96,<br>97, 98, 99])<br>15. array([[[55, 56, 57, 58, 59],<br>[65, 66, 67, 68, 69],<br>[75, 76, 77, 78, 79],<br>[85, 86, 87, 88, 89],<br>[95, 96, 97, 98, 99]],<br>[[50, 51, 52, 53, 54],<br>[60, 61, 62, 63, 64],<br>[70, 71, 72, 73, 74],<br>[80, 81, 82, 83, 84],<br>[90, 91, 92, 93, 94]]]) |
| **INDEXING AND SLICING**<br>1. array1D<br>2. array1D[0]<br>3. array1D[-1]<br>4. array1D[::5]<br>5. array1D[::-10]<br>6. array2D<br>7. array2D[0:2] | 1. array([50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,<br>66,67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83<br>,84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])<br>2. 50<br>3. 99<br>4. array([50, 55, 60, 65, 70, 75, 80, 85, 90, 95])<br>5. array([99, 89, 79, 69, 59])<br>6. array([[50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,64,65,<br>66, 67, 68, 69, 70, 71, 72, 73, 74],<br>[75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,91, 9 |

| | |
|---|---|
| 8. array2D[0:2 , 1]<br>9. array2D[0:2][1]<br>10. array3D<br>11. array3D[0:2]<br>12. array3D[0:2 , 1]<br>13. array3D[0:2][1] | 2, 93, 94, 95, 96, 97, 98, 99]])<br>7. 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],<br>[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]])<br>8. array([51, 76])<br>9. 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])<br>10. array([[[50, 51, 52, 53, 54],<br>[55, 56, 57, 58, 59]],<br>[[60, 61, 62, 63, 64],<br>[65, 66, 67, 68, 69]],<br>[[70, 71, 72, 73, 74],<br>[75, 76, 77, 78, 79]],<br>[[80, 81, 82, 83, 84],<br>[85, 86, 87, 88, 89]],<br>[[90, 91, 92, 93, 94],<br>[95, 96, 97, 98, 99]]])<br>11. array([[[50, 51, 52, 53, 54],<br>[55, 56, 57, 58, 59]],<br>[[60, 61, 62, 63, 64],<br>[65, 66, 67, 68, 69]]])<br>12. array([[55, 56, 57, 58, 59],<br>[65, 66, 67, 68, 69]])<br>13. array([[60, 61, 62, 63, 64],<br>[65, 66, 67, 68, 69]]) |
| **STATISTICAL FUNCTIONS**<br><br>1. max -----> maximum in array<br>>>> array1D.max()<br>2. max -----> maximum in array via axis = 1<br>>>> array2D.max(axis = 1)<br>3. max -----> maximum in array via axis = 0<br>>>> array2D.max(axis = 0)<br>4. min -----> minimum in array<br>>>> array1D.min()<br>5. min -----> minimum in array via axis = 1<br>>>> array2D.min(axis = 1)<br>6. min -----> minimum in array via axis = 0<br>>>> array2D.min(axis = 0)<br>7. sum -----> sum of all elements<br>>>> array1D.sum()<br>8. sum -----> sum of elements in array via axis = 1<br>>>> array2D.sum(axis = 1)<br>9. sum -----> sum of elements in array via axis = 0<br>>>> array2D.sum(axis = 0)<br>10. mean -----> sum of all elements divided by size of elements<br>>>> array1D.mean()<br>11. median -----> average of the middle elements<br>>>> np.median(array1D)<br>12. varience -----> sum of (x-u)**2 divided by n<br>>>> np.var(array1D)<br>13. standard deviation -----> sqrt of (sum of (x-u)**2 divided by n)<br>>>> np.std(array1D) | 1. 99<br>2. array([74, 99])<br>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])<br>4. 50<br>5. array([50, 75])<br>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])<br>7. 3725<br>8. array([1550, 2175])<br>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])<br>10. 74.5<br>11. 74.5<br>12. 208.25<br>13. 14.430869689661812 |
| **STACKING and SPLITTING**<br><br>1. vstack -----> vertical attachment (columns are equal in 2 arrays)<br>>>> np.vstack((array1,array2))<br>2. hstack -----> horizontal attachment (rows are equal in 2 arrays)<br>>>> np.hstack((array1,array2))<br>3. dstack -----> parllel elements in 2 arrays along the third axis (min 3D array is required)<br>>>> np.dstack((array1,array2))<br>4. hsplit -----> horizontal split means dividing columns | 1. array([[ 1, 2, 3],<br>[ 4, 5, 6],<br>[ 7, 8, 9],<br>[11, 12, 13],<br>[14, 15, 16]])<br>2. array([[ 1, 2, 3, 11, 12, 13],<br>[ 4, 5, 6, 14, 15, 16],<br>[ 7, 8, 9, 17, 18, 19]])<br>3. array([[[ 1, 11],<br>[ 2, 12],<br>[ 3, 13]]]) |

| | |
|---|---|
| >>> np.hsplit(array2D,5)<br>5. vsplit -----> vertical split means dividing rows<br>>>> np.vsplit(array2D,1)<br>6. where -----> returns values with some conditions<br>>>> np.where(array2D%5==0, array2D , 'False') | 4. [array([[50, 51, 52, 53, 54],<br>[75, 76, 77, 78, 79]]),<br>array([[55, 56, 57, 58, 59],<br>[80, 81, 82, 83, 84]]),<br>array([[60, 61, 62, 63, 64],<br>[85, 86, 87, 88, 89]]),<br>array([[65, 66, 67, 68, 69],<br>[90, 91, 92, 93, 94]]),<br>array([[70, 71, 72, 73, 74],<br>[95, 96, 97, 98, 99]])]<br>5. [array([[50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64,<br>65,66, 67, 68, 69, 70, 71, 72, 73, 74],<br>[75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,<br>91, 92, 93, 94, 95, 96, 97, 98, 99]])]<br>6. array([['50', 'False', 'False', 'False', 'False', '55', 'False', 'False',<br>'False', 'False', '60', 'False', 'False', 'False', 'False', '65',<br>'False', 'False', 'False', 'False', '70', 'False', 'False',<br>'False', 'False'],<br>['75', 'False', 'False', 'False', 'False', '80', 'False', 'False',<br>'False', 'False', '85', 'False', 'False', 'False', 'False', '90',<br>'False', 'False', 'False', 'False', '95', 'False', 'False',<br>'False', 'False']], dtype='<U11') |
| **SET OPERATIONS**<br><br>1. union1d -----> combining<br>>>> np.union1d(array1,array2)<br>2. intersect1d -----> common<br>>>> np.intersect1d(array1,array2)<br>3. setdiff1d -----> subtracting array2 from array1 and printing remaining elements in array1<br>>>> np.setdiff1d(array1,array2) | 1. array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 11, 12, 13, 14, 15, 16, 17, 18,<br>19])<br>2. array([], dtype=int32)<br>3. array([1, 2, 3, 4, 5, 6, 7, 8, 9]) |
| **ARITHMETIC OPERATIONS**<br><br>array1 = np.array([[1,2,3],[4,5,6],[7,8,9]])<br>array2 = np.array([[11,12,13],[14,15,16],[17,18,19]])<br>1. array1 + array2<br>2. array1 - array2<br>3. array1 * array2<br>4. array1 / array2<br>5. array1 // array2<br>6. array1 % array2<br>7. array1 @ array2 | 1.array([[12, 14, 16],<br>[18, 20, 22],<br>[24, 26, 28]])<br>2.array([[-10, -10, -10],<br>[-10, -10, -10],<br>[-10, -10, -10]])<br>3.array([[ 11,  24,  39],<br>[ 56,  75,  96],<br>[119, 144, 171]])<br>4.array([[0.09090909, 0.16666667, 0.23076923],<br>[0.28571429, 0.33333333, 0.375     ],<br>[0.41176471, 0.44444444, 0.47368421]])<br>5.array([[0, 0, 0],<br>[0, 0, 0],<br>[0, 0, 0]])<br>6.array([[1, 2, 3],<br>[4, 5, 6],<br>[7, 8, 9]])<br>7.array([[ 90,  96, 102],<br>[216, 231, 246],<br>[342, 366, 390]]) |
| **AGGREGATE FUNCTIONS**<br><br>a1 = np.array([[1,2,3],[4,5,6],[7,8,9]])<br>a2 = np.array([[11,12,13],[14,15,16],[17,18,19]])<br>1. np.add(a1,a2)<br>2. np.subtract(a1,a2)<br>3. np.multiply(a1,a2)<br>4. np.matmul(a1,a2) | 1.array([[12, 14, 16],<br>[18, 20, 22],<br>[24, 26, 28]])<br>2.array([[-10, -10, -10],<br>[-10, -10, -10],<br>[-10, -10, -10]])<br>3.array([[ 11,  24,  39],<br>[ 56,  75,  96],<br>[119, 144, 171]])<br>4.array([[ 90,  96, 102],<br>[216, 231, 246],<br>[342, 366, 390]]) |
| **APPEND AND CONCATENATE**<br><br>1. append -----> adding at the last of the array<br>2. concatenate -----> adding in left-right or up-down direction<br>a1 = np.array([[1,2,3],[4,5,6],[7,8,9]]) | 1. array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 11, 12, 13, 14, 15, 16, 17, 18,<br>19])<br>2. array([[ 1,  2,  3],<br>[ 4,  5,  6],<br>[ 7,  8,  9], |

| | |
|---|---|
| a2 = np.array([[11,12,13],[14,15,16],[17,18,19]])<br>1. np.append(a1,a2)<br>np.append(a1,a2,axis = 1)<br>np.append(a1,a2,axis = 0)<br>2. np.concatenate((a1,a2))<br>np.concatenate((a1,a2) , axis = 1)<br>np.concatenate((a1,a2) , axis = 0) | [11, 12, 13],<br>[14, 15, 16],<br>[17, 18, 19]]) |
| **FILTERING ARRAYS**<br><br>a1 = np.array([[1,2,3],[4,5,6],[7,8,9]])<br>a2 = np.array([[11,12,13],[14,15,16],[17,18,19]])<br>1. a1 > 5<br>2. a2 < 15 | 1. array([[False, False, False],<br>[False, False,  True],<br>[ True,  True,  True]])<br>2. array([[ True,  True,  True],<br>[ True, False, False],<br>[False, False, False]]) |
| **Numpy RANDOM NUMBERS**<br><br>1. np.random.rand -----> generates an array with random numbers that are uniformly distributed between 0 and 1<br>>>> array2D = np.random.rand(2,3)<br>2. np.random.randn -----> generates an array with random numbers that are normally distributed with mean = 0 and sd = 1<br>>>> array2D = np.random.randn(2,3)<br>3. np.random.randint -----> generates an array with random numbers that are uniformly distributed between 0 and given integer<br>>>> array2D = np.random.randint(50 , size = (2,3))<br>4. np.random.uniform -----> generates array with random numbers that are uniformly distributed within the given range of values<br>>>> array2D = np.random.uniform(50,60 , size = (2,3))<br>5. np.random.seed -----> puts the random values constant even though we execute the random code for multiple times<br>>>> np.random.seed(1372)<br>>>> array2D = np.random.randint(50 , size = (2,3)) | 1. array([[0.73852266, 0.13648889, 0.53915898],<br>[0.52860591, 0.01443914, 0.09355033]])<br>2. array([[-0.72506201,  0.06380681, -1.17170102],<br>[-0.29041479, -0.80791029,  0.75431523]])<br>3. array([[22, 49, 34],<br>[48, 35, 36]])<br>4. array([[58.08107549, 59.19999339, 50.4455265 ],<br>[59.53420965, 57.62848807, 52.49294163]])<br>5. array([[20,  2, 42],<br>[ 7, 25, 46]]) |
| **EXPANDING AND SQUEEZING**<br><br>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<br>a1 = np.array([1,2,3,4,5,6,7,8,9])<br>a1<br>a1.ndim<br>a2 = np.expand_dims(a1 , axis = 0)<br>a2<br>a2.ndim<br>a2 = np.expand_dims(a1 , axis = 1)<br>a2<br>a2.ndim<br>2. squeeze() -----> removes the axis that has a single entry<br>a1 = np.array([[[1,2,3,4,5,6,7,8,9]]])<br>a1<br>a1.ndim<br>a2 = np.squeeze(a1 , axis = 0)<br>a2<br>a2.ndim<br>a2 = np.squeeze(a1 , axis = 1)<br>a2<br>a2.ndim | **#expanding**<br>array([1, 2, 3, 4, 5, 6, 7, 8, 9])<br>1<br>array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])<br>2<br>array([[1],<br>[2],<br>[3],<br>[4],<br>[5],<br>[6],<br>[7],<br>[8],<br>[9]])<br>2<br>**#squeezing**<br>array([[[1, 2, 3, 4, 5, 6, 7, 8, 9]]])<br>3<br>array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])<br>2<br>array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])<br>2 |