Done till numpy 7

np.array([ ] , dtype =)

np.arange(start , end , step)

end is not included

np.reshape(array , shape(x , y)) :

product of x and y should be equal to total elements inside array

np.ones((x,y) , dtype =)

x : row , y = col

np.zeros((x,y),dtype =)

x : row , y = col

np.identity(n)

nxn identity matrix

np.full(size , element to fill)

np.random.random((x,y))

x : row , y = col , random val 0-1

np.random.randint(start , end , (x,y))

x : row , y = col ,end not included

np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=float)

By default num = 50

Means between start and end 50 values will be generated on even spaces

Endpoint = True means include end value

np.ndim(array)

Tells the dimensions of the array

np.shape(array)
returns(row , col) of array

np.size(array)
Returns the length of array

Array.dtype
Returns the dtype of array

array.astype(data_type)
Creates a copy of array with given data type

Structured array

My_data_type = [(‘name’:’U12’),(‘Age’:’i4’),(‘Marks,’f4’)]

np.array([(‘Jibrna’ ,21 ,57.5)])
U12 : Unicode string of length 12 , You may changed it
i4 : Integers of 4 bytes , 32 bits (int32)
f4 : floating point og 4 bytes , 32 bits (float32)

Slicing

Array[row_start:row_end , col_start:col_end]
End is not included

Looping

for element in np.nditer(array, flags=[], op_flags=[], order='K'):

flags=[external_loops]: iterate over arrays not on elements in n dimensional array (not 1 dimension)

op_flags= [readwrite] : to modify the elements of array
i[...] = i*10 , this [...] is part of syntax
Order = ‘k’ : row , ‘F’ : columns , by default k

np.ravel(array)
flats the n-d array

np.transpose(array)

STACKING

In NumPy, stacking refers to combining multiple arrays along a new axis (row-wise, column-wise, or depth-wise) to create a larger array.

np.hstack((array1, array2))
horizontal stack : rows same , col change
np.vstack((array1 , array2))
Vertical stack : rows change , col same

```
if x :   [2 x 2]  :  [2 x 2 ]  =  [2 x 4]
if y :   [2 x 2]  :  [2 x 2 ]  =  [4 x 2]
```

In NumPy, concatenation refers to combining two or more arrays along a specified axis (0 for rows, 1 for columns, etc.). This can be done using the function np.concatenate(), which is more general and flexible compared to other stacking functions like np.vstack() and np.hstack().

np.concatenate((array1,array2) , axis = 1/0)

- **axis=0: Operates along rows (vertically). For example, `np.sum(arr_2d, axis=0)` calculates the sum of each column.**
- **axis=1: Operates along columns (horizontally). `np.sum(arr_2d, axis=1)` calculates the sum of each row.**

Axis:
[Understanding Axes in NumPy. Your Key to Array Manipulation | by Dagang Wei | Medium]

np.expand_dims(array , axis = 0/1): video check
np.cumsum(array , axis = 0)
np.cumproduct(array , axis = 0/1)
np.percentile(array ,percentile_rank)
np.hist(array , bins = [ ])
np.sort(array , axis =)
By default axis = 0 : col wise

np.sort(array[])[::-1]
Sort in descending order
np.append(array2 , new_col/new_row , axis = 0/1)

np.unique(array)

The correlation coefficient, often denoted as r. It measures the strength and direction of a linear relationship between two variables. It provides a value between -1 and +1: +1: Perfect positive correlation — as one variable increases, the other also increases in a perfectly linear manner. -1: Perfect negative correlation — as one variable increases, the other decreases in a perfectly linear manner. 0: No correlation — there is no linear relationship between the two variables.

np.corrcoef(x,y)

np.isin(jo dekhni , game dekni)
example:
items = np.array([10,20,30,40,50,60,70,80,100])
main_array = np.array([10,30,60,100])
np.isin(main_array,items)

array([True, True, True, True])

np.union1d(array1,array2)

np.intersection1d(array1,array2)

np.setdiff1d(array1,array2)

Wo items jo srf phle set m hen dosre m nh

np.setxor1d(array1,arary2)

remove common from both arrays

```
# NumPy, argmax is a function that returns the index of the maximum value
along a specified axis or in the entire array.
# It is commonly used to find the position of the largest element in an array
```

np.argmin(array)
np.argmax(array)

np.where(condtion , on True:x , on false:y)

numpy.where(condition, [x, y]) # condition: An array-like object (e.g., a NumPy array or a condition) that returns True or False for each element. # x: Values to select when the condition is True. # y: Values to select when the condition is False. # The x and y parameters are optional. If only the condition parameter is provided, numpy.where will return the indices of the elements where the condition is True

.

a = np.arange(10)

np.where(a>4)

returns only the indices where value is greater than 4 np.where(a>4 , "it is greater than 4","it is less than 4") np.where((a>5) & (a<10) , "It is between 5 and 10 ","Out of range")

WHEN FOR AND CONDTION USE & # No, arrays are not scalars. In NumPy, scalars refer to single values (e.g., a single number), while arrays are collections of values. When you're working with arrays and need to combine multiple conditions, you must use the bitwise operators like & (AND), | (OR), and ~ (NOT), # rather than the logical operators like and, or, and not which are meant for scalar (single value) operations. # and, or, and not are logical operators used to combine boolean values in Python. These operators work with single boolean values (i.e., scalars), not with arrays.