Numpy Cheat Sheet

Section 1

Import Convention

Import numpy as np	Abbreviate numpy as np

A-dimensional Array

Syntax	np.array()
1D	np.array([1,2,3,4]) Output: array([1, 2, 3, 4])
2D	np.array([[1,2,3], [4,5,6]]) Output: array([[1, 2, 3], [4, 5, 6]])
3D	np.array([[[1,2,3], [4,5,6]]]) Output: array([[[1, 2, 3], [4, 5, 6]]])
4D	np.array([[[[1,2,3]]]]) Output: array([[[[1, 2, 3]]]])

Creating Array

a = np.array([1,3,4,8]) Output: [1 3 4 8]
b = np.array([1.4, 3.2, 5], dtype = float) Output: [1.4 3.2 5.]
c = np.array([1,3,5,7,9,11], dtype = int64) Output: [1 3 5 7 9 11]

Initial Placeholders

Create an array with zeros.	np.zeros((3,8)) Output: array([[0., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 0.]])
Create an array with ones.	np.ones((3,8)) Output: array([[1., 1., 1., 1., 1., 1., 1., 1.], [1., 1., 1., 1., 1., 1., 1., 1.], [1., 1., 1., 1., 1., 1., 1., 1.]])
Create an array with even-spaced step values.	np.arange(1, 30, 5) Output: array([1, 6, 11, 16, 21, 26])
Create an evenly-spaced array with a specified number of values.	np.linspace(0,4,8) Output: array([0. , 0.57142857, 1.14285714, 1.71428571, 2.28571429, 2.85714286, 3.42857143, 4.])
Create an entire array with a constant value.	np.full((4,4),9) Output: array([[9, 9, 9, 9], [9, 9, 9, 9], [9, 9, 9, 9], [9, 9, 9, 9]])
Create an identity matrix.	np.identity(3) Output: array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])
Create an identity matrix with m*n dimension.	np.eye(3,5) Output: array([[1., 0., 0., 0., 0.], [0., 1., 0., 0., 0.], [0., 0., 1., 0., 0.]])
Create an identity matrix with the m*n dimension with step shift.	np.eye(3, k=1) Output: array([[0., 1., 0.], [0., 0., 1.], [0., 0., 0.]])
Create a diagonal matrix.	np.diag([1,1,1]) Output: array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
Create an array with random integers.	np.random.randint(2,8,3) Output: array([2, 6, 3])
Create an array between [0, 1).	np.random.rand(5,2) Output: array([[0.52620975, 0.27200853], [0.04753095, 0.38419669], [0.29254718, 0.66309665], [0.09115936, 0.62305064], [0.7984203 , 0.39769068]])
Create an array with a mean 0 and a standard deviation of 1.	np.random.randn(2,3) Output: array([[-0.98863765, 0.33808866, 0.07083797], [-0.58465781, -1.53241981, -1.03018067]])
Create a random array in the [0.0, 1.0) interval	np.random.random_sample(10) Output: array([0.33360613, 0.54360933, 0.61117749, 0.52978991, 0.70083391, 0.61659811, 0.87412798, 0.62451739, 0.03659576, 0.12249504]) np.random.random(10) Output: array([0.79213708, 0.65407051, 0.70394522, 0.72384937, 0.80480269, 0.51195845, 0.00852462, 0.571088 , 0.59704964, 0.74560852])

Section 2

Saving and Loading

Saving and Loading on Disk

a = np.array([[1,2,3], [4,5,6]]) np.save('arr' ,a)
b = np.savez('arr', a) #saving array
c =np.load('arr') #loading array print(c)

Saving and Loading text/csv Files

np.loadtxt('New_file.txt') #From a text file
np.savetxt('New_file.txt',arr,delimiter=' ') #Writes to a text file
np.genfromtxt('New_file.csv',delimiter=',') #From a CSV file
np.savetxt('New_file.csv',arr,delimiter=',') #Writes to a CSV file

Properties

arr1 = np.array([[1,2,3], [4,5,6]])

Array dimensions	arr1.ndim Output: 2
Length of array	len(arr1) Output: 2
Shape of the array	arr1.shape Output: (2,3)
Name of datatype	arr1.dtype.name Output: 'int'
Type-cast array	arr1.astype(int) Output: array([[1, 2, 3], [4, 5, 6]])
Data type of an array	arr1.dtype Output: dtype('int64')

Indexing and Slicing, Subsetting

- **Boolean Indexing**
- **Fancy Indexing**
- **Subsetting**
- **Slicing**

Indexing	#forward Index arr= ['a', 'b', 'c', 'd', 'e'] index→ 0 , 1 , 2, 3, 4 #backward indexing arr= ['a', 'b', 'c', 'd', 'e'] index→ -5 , -4 , -3, -2, -1
Boolean Indexing	arr1 = np.array([[1,2,3], [4,5,6]]) arr1[arr1 > 3] #Selects elements from an array based on boolean condition Output: array([4, 5, 6])
Fancy Indexing	arr = np.array([0, 10, 20, 30, 40, 50]) indices = np.array([0, 1, 5]) #indexing using fancy indexing fancy_ind = arr[indices] print(fancy_ind) # Output: [0 10 50] Output: [0 10 50]
Subset	a = arr1[1] #selects the second element Output: [4 5 6]
Slicing	arr1[0:3] #selecting elements from 0-3 Output: array([[1, 2, 3], [4, 5, 6]])

Array Mathematics

#Create arrays	arr1 = np.array([1, 2, 3]) arr2 = np.array([4, 5, 6]) Output: [1 2 3] Output: [4 5 6]

Arithmetic Function

Addition	np.add(arr1, arr2) Output: array([5, 7, 9])
Subtraction	np.subtract(arr1, arr2) Output: array([-3, -3, -3])
Multiplication	np.multiply(arr1, arr2) Output: array([4, 10, 18])
Division	np.divide(arr1,arr2) Output: array([0.25, 0.4 , 0.5])
Exponentiation	np.power(arr1, arr2) Output: array([1, 32, 729])
Floor/Integer Division	np.floor_divide(arr1, arr2) Output: array([0, 0, 0])
Modulus	np.mod(arr1, arr2) Output: array([1, 2, 3])
Square Root	np.sqrt(arr1) Output: array([1. , 1.41421356, 1.73205081])
Unary Negation	np.negative(arr1, arr2) Output: array([-1, -2, -3])

Mathematical Functions

Trigonometric Functions	np.sin(arr1) Output: array([0.84147098, 0.90929743, 0.14112001]) np.cos(arr1) Output: array([0.54030231, -0.41614684, -0.9899925]) np.tan(arr1) Output: array([1.55740772, -2.18503986, -0.14254654])
Exponential and Logarithmic Functions	np.exp(arr1) Output: array([2.71828183, 7.3890561 , 20.08553692]) np.log(arr1) Output: array([0. , 0.69314718, 1.09861229])
Statistical Functions	np.mean(arr1) #Finding mean Output: 2.0 np.std(arr1)#Finding standard deviation Output: 0.816496580927726 np.median(arr1) #finding median Output: 2.0

Comparison and Logical Operator

np.equal(arr1, arr2) Output: [False False False] np.not_equal(arr1, arr2) Output: [True True True] np.greater(arr1, arr2) Output: [False False False]
np.logical_and(arr1, arr2) Output: [True True True] np.logical_or(arr1,arr2) Output: [True True True] np.logical_not(arr1) Output: [False False False]

Aggregation Functions

arr1 = np.array([1,2,3]) arr2 =np.array([4,5,6]) np.sum(arr1 + arr2) #Summation Output: 21
np.corrcoef(arr1) Output: 1.0
np.max(np.concatenate(arr1, arr2)) #max of two Output: 6 np.min(np.concatenate(arr1, arr2)) #min of two Output: 1
np.cumsum(arr1) #cumulative sum Output: [1 3 6]
np.cumprod(arr1) #cumulative multiplication Output: array([1, 2, 6])

Section 3

Data Types

signed 64-bit int type	np.int64
Double standard float point	np.float32
Complex numbers	np.complex
Boolean data type	np.bool
Object type	np.object
Fixed-length string	np.string_
Unicode	np.unicode_

Arithmetic Function

Addition	np.add(arr1, arr2) Output: array([5, 7, 9])
Subtraction	np.subtract(arr1, arr2) Output: array([-3, -3, -3])
Multiplication	np.multiply(arr1, arr2) Output: array([4, 10, 18])
Division	np.divide(arr1,arr2) Output: array([0.25, 0.4 , 0.5])
Exponentiation	np.power(arr1, arr2) Output: array([1, 32, 729])
Floor/Integer Division	np.floor_divide(arr1, arr2) Output: array([0, 0, 0])
Modulus	np.mod(arr1, arr2) Output: array([1, 2, 3])
Square Root	np.sqrt(arr1) Output: array([1. , 1.41421356, 1.73205081])
Unary Negation	np.negative(arr1, arr2) Output: array([-1, -2, -3])

Mathematical Functions

Trigonometric Functions	np.sin(arr1) Output: array([0.84147098, 0.90929743, 0.14112001]) np.cos(arr1) Output: array([0.54030231, -0.41614684, -0.9899925]) np.tan(arr1) Output: array([1.55740772, -2.18503986, -0.14254654])
Exponential and Logarithmic Functions	np.exp(arr1) Output: array([2.71828183, 7.3890561 , 20.08553692]) np.log(arr1) Output: array([0. , 0.69314718, 1.09861229])
Statistical Functions	np.mean(arr1) #Finding mean Output: 2.0 np.std(arr1)#Finding standard deviation Output: 0.816496580927726 np.median(arr1) #finding median Output: 2.0

Comparison and Logical Operator

np.equal(arr1, arr2) Output: [False False False] np.not_equal(arr1, arr2) Output: [True True True] np.greater(arr1, arr2) Output: [False False False]
np.logical_and(arr1, arr2) Output: [True True True] np.logical_or(arr1,arr2) Output: [True True True] np.logical_not(arr1) Output: [False False False]

Aggregation Functions

arr1 = np.array([1,2,3]) arr2 =np.array([4,5,6]) np.sum(arr1 + arr2) #Summation Output: 21
np.corrcoef(arr1) Output: 1.0
np.max(np.concatenate(arr1, arr2)) #max of two Output: 6 np.min(np.concatenate(arr1, arr2)) #min of two Output: 1
np.cumsum(arr1) #cumulative sum Output: [1 3 6]
np.cumprod(arr1) #cumulative multiplication Output: array([1, 2, 6])

Functions

- **Updating array elements**
- **Removing array elements**
- **Copy array**
- **Return condition specific element**
- **Find unique element**
- **Sorting**

Updating array elements	arr = [11,12,15,17,19] #update 12 with 13 arr[1]= 13 arr Output: [11,13,15,17,19]
Removing array elements	np.delete(arr, 2) #removes element at index 2 Output: [11,13,17,19]
Copying arrays	arr.view() #create view of the array with the same data np.copy(arr1) #creates a copy of array arr1.copy() #creates deep copy of array
Return a condition-specific element.	ele = arr[arr > 3] # return elements greater than 3 Output: [11 12 15 17 19]
Find a unique element	np.unique(arr) #returns unique elements Output: array([11, 12, 15, 17, 19])
Sorting	#ascending order np.sort(arr) Output: array([11, 12, 15, 17, 19]) #descending order np.sort(arr)[::-1] Output: array([19, 17, 15, 12, 11])

Array Manipulation

- **Reshaping array**
- **Flattening array**
- **Transposing array**
- **Concatenating arrays vertically and horizontally**
- **Splitting array**
- **Flip array**

Reshaping Array	arr.reshape(3,4) #changes to 3*4 matrix Output: array([[1, 4, 7, 9], [2, 4, 7, 9], [2, 3, 4, 5]])
Flattening Array	arr.flatten() Output: array([1, 4, 7, 9, 2, 4, 7, 9, 2, 3, 4, 5]) arr.ravel() Output: array([1, 4, 7, 9, 2, 4, 7, 9, 2, 3, 4, 5])
Transposing Array	arr.T Output: array([[1, 2, 2], [4, 4, 3], [7, 7, 4], [9, 9, 5]])
Concatenating Arrays	#vertical np.vstack((arr1, arr2)) Output: array([[1, 2, 3], [4, 5, 6]]) #horizontal np.hstack((arr1, arr2)) Output: array([1, 2, 3, 4, 5, 6])
Splitting Array	#splitting array into n sub-arrays np.split(arr, 3) Output: [array([[1, 4, 7, 9]]), array([[2, 4, 7, 9]]), array([[2, 3, 4, 5]])]
Flip Array	np.flip(arr, axis=0) #vertical flip np.flip(arr, axis=1) #horizontal flip

Linear Algebra

- **Matrix Operations**
- **Matrix Properties**
- **Solving Equations**
- **Eigenvalues and Eigenvectors**

Matrix Operations

c = np.add(a, b) # Matrix addition
d = np.subtract(a, b) # Matrix subtraction
e = np.dot(a, b) # Matrix multiplication
f = np.divide(a, b) # Matrix division

Matrix Properties

det_a = np.linalg.det(a) # Determinant
a_transpose = np.transpose(a) # Transpose
a_inv = np.linalg.inv(a)# Inverse

Solving Equations

Solving linear equations x = np.linalg.solve(a, b)

Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors eigenvalues, eigenvectors = np.linalg.eig(a)

Help in Numpy

Help in NumPy

np.info(np.ndarray.dtype) help(np.ndarray.dtype)