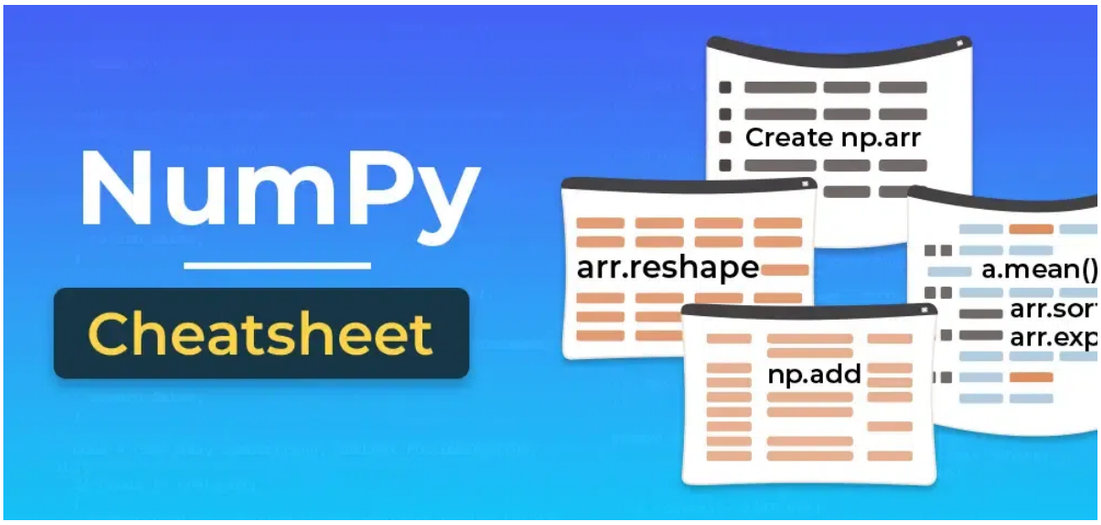
# NumPy Cheat Sheet: Beginner to Advanced (PDF)

Last Updated : 18 Mar, 2024

NumPy stands for Numerical Python. It is one of the most important foundational packages for numerical computing & data analysis in Python. Most computational packages providing scientific functionality use NumPy's array objects as the lingua franca for data exchange.

In this Numpy Cheat sheet for Data Analysis, we've covered the basics to advanced functions of Numpy including creating arrays, Inspecting properties as well as file handling, Manipulation of arrays, Mathematics Operations in Array and more with proper examples and output. By the end of this Numpy cheat sheet, you will gain a fundamental comprehension of NumPy and its application in Python for data analysis.




*NumPy Cheat Sheet*

## What is NumPy?

NumPy was initially created by Travis Oliphant in 2005 as an open-source project. NumPy is a powerful Python library that provides support for large, multi-dimensional arrays and matrices, along with a wide collection of mathematical functions to operate on these arrays. It is an essential tool for scientific computing and data analysis in Python.

Numpy CheatSheat PDF Download (Free).

Table of Content:

# NumPy Cheat Sheet 2023

## 1. Creating Arrays Commands

Arrays in NumPy are of fixed size and homogeneous in nature. They are faster and more efficient because they are written in C language and are stored in a continuous memory location which makes them easier to manipulate. NumPy arrays provide N-dimensional array objects that are used in linear algebra, Fourier Transformation, and random number capabilities. These array objects are much faster and more efficient than the Python Lists.

### Creating One Dimensional Array

NumPy one-dimensional arrays are a type of linear array. We can create a NumPy array from Python List, Tuple, and using fromiter() function.

| Creating One Dimensional Array | Example |
|---|---|
| From Python List | np.array([1, 2, 3, 4, 5]) |
| From Python Tuple | np.array((1, 2, 3, 4, 5)) |
| fromiter() function | np.fromiter((a for a in range(8)), float) |

### Python3

```
# create a NumPy array from a list
li = [1, 2, 3, 4]
print(np.array(li))

# create a NumPy array from a tuple
tup = (5, 6, 7, 8)
print(np.array(tup))

# create a NumPy array using fromiter()
iterable = (a for a in range(8))
print(np.fromiter(iterable, float))
```

Output:

```
[1 2 3 4]
[5 6 7 8]
[0. 1. 2. 3. 4. 5. 6. 7.]
```

### Creating Multi-Dimensional Array

Numpy multi-dimensional arrays are stored in a tabular form, i.e., in the form of rows and columns.

| Create Two Dimensional Array | Example |
|---|---|
| Using Python Lists | np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]) |
| Using empty() | np.empty([4, 3], dtype=int) |

## Python3

```
# create a NumPy array from a list
list_1 = [1, 2, 3, 4]
list_2 = [5, 6, 7, 8]
list_3 = [9, 10, 11, 12]
print(np.array([list_1, list_2, list_3]))

# create a NumPy array using numpy.empty()
print(np.empty([4, 3], dtype=int))
```

Output:

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

## 2. Initial Placeholders

### Example 1: For 1-Dimensional NumPy Arrays

Initial placeholders for a Numpy 1-dimension array can be created by using various Numpy functions.

| Initial Placeholders for 1D Array | Example |
|---|---|
| arange() | np.arange(1, 10) |

| Initial Placeholders for 1D Array | Example |
|---|---|
| linespace() | np.linspace(1, 10, 3) |
| zeros() | np.zeros(5, dtype=int) |
| ones() | np.ones(5, dtype=int) |
| random.rand() | np.random.rand(5) |
| random.randint() | np.random.randint(5, size=10) |

Python3

```
# create a NumPy array using numpy.arange()
print(np.arange(1, 10))

# create a NumPy array using numpy.linspace()
print(np.linspace(1, 10, 3))

# create a NumPy array using numpy.zeros()
print(np.zeros(5, dtype=int))

# create a NumPy array using numpy.ones()
print(np.ones(5, dtype=int))

# create a NumPy array using numpy.random.rand()
print(np.random.rand(5))

# create a NumPy array using numpy.random.randint()
print(np.random.randint(5, size=10))
```

Output:

```
[1 2 3 4 5 6 7 8 9]
[ 1.   5.5 10. ]
[0 0 0 0 0]
[1 1 1 1 1]
[0.31447226 0.89090771 0.45908938 0.92006507 0.37757036]
[4 3 2 3 1 2 4 1 4 2]
```

Example 2: For N-dimensional Numpy Arrays

Initial placeholders for Numpy two dimension arrays can be created by using various NumPy functions.

| Initial Placeholders for 2D Array | Example |
|---|---|
| zeros() | np.zeros([4, 3], dtype = np.int32) |

| Initial Placeholders for 2D Array | Example |
|---|---|
| ones() | np.ones([4, 3], dtype = np.int32) |
| full() | np.full([2, 2], 67, dtype = int) |
| eye() | np.eye(4) |

Python3

```
# create a NumPy array using numpy.zeros()
print(np.zeros([4, 3], dtype = np.int32))

# create a NumPy array using numpy.ones()
print(np.ones([4, 3], dtype = np.int32))

# create a NumPy array using numpy.full()
print(np.full([2, 2], 67, dtype = int))

# create a NumPy array using numpy.eye()
print(np.eye(4))
```

Output:

```
 [[0 0 0]
  [0 0 0]
  [0 0 0]
  [0 0 0]]
 [[1 1 1]
  [1 1 1]
  [1 1 1]
  [1 1 1]]
 [[67 67]
  [67 67]]
 [[1. 0. 0. 0.]
  [0. 1. 0. 0.]
  [0. 0. 1. 0.]
  [0. 0. 0. 1.]]
```

## 3. Inspecting Properties

NumPy arrays possess some basic properties that can be used to get information about the array such as the size, length, shape, and datatype of the array. Numpy arrays can also be converted to a list and be change their datatype.

| Inspecting Properties | Example |
|---|---|
| Size | arr.size |

| Inspecting Properties | Example |
|---|---|
| Length | len(arr) |
| Shape | arr.shape |
| Datatype | arr.dtype |
| Changing Datatype of Array | arr.astype('float64') |
| Converting Array to List | arr.tolist() |

Example 1: One Dimensional Numpy Array

Python3

```
arr = np.asarray([1, 2, 3, 4])
# check size of the array
print("Size:", arr.size)

# check len of the array
print("len:", len(arr))

# check shape of the array
print("Shape:", arr.shape)

# check dtype of the array elements
print("Datatype:", arr.dtype)

# change the dtype to 'float64'
arr = arr.astype('float64')
print(arr)
print("Datatype:", arr.dtype)

# convert array to list
lis = arr.tolist()
print("\nList:", lis)
print(type(lis))
```

Output:

```
Size: 4
len: 4
Shape: (4,)
Datatype: int64
[1. 2. 3. 4.]
Datatype: float64
List: [1.0, 2.0, 3.0, 4.0]
<class 'list'>
```

Example 2: N-Dimensional Numpy Array

Python3

```
# Two dimensional numpy array
list_1 = [1, 2, 3, 4]
list_2 = [5, 6, 7, 8]
list_3 = [9, 10, 11, 12]
arr = np.array([list_1, list_2, list_3])

# check size of the array
print("Size:", arr.size)

# check length of the array
print("Length:", len(arr))

# check shape of the array
print("Shape:", arr.shape)

# check dtype of the array elements
print("Datatype:", arr.dtype)

# change the dtype to 'float64'
arr = arr.astype('float64')
print(arr)
print(arr.dtype)

# convert array to list
lis = arr.tolist()
print("\nList:", lis)
print(type(lis))
```

Output:

```
Size: 12
Length: 3
Shape: (3, 4)
Datatype: int64
[[ 1.  2.  3.  4.]
 [ 5.  6.  7.  8.]
 [ 9. 10. 11. 12.]]
float64
List: [[1.0, 2.0, 3.0, 4.0], [5.0, 6.0, 7.0, 8.0], [9.0, 10.0, 11.0, 12.0]]
<class 'list'>
```

### Getting Information on a Numpy Function

The np.info() function is used to get information about any Numpy function, class, or module along with its parameters, return values, and any other information.

#### Python3

```
import sys
print(np.info(np.add, output=sys.stdout))
```

Output:

```
add(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K',
dtype=None, subok=True[, signature, extobj])
Add arguments element-wise.
Parameters
----------
```

```
x1, x2 : array_like
    The arrays to be added.
.....
```

## 4. Saving and Loading File

Numpy arrays can be stored or loaded from a disk file with the '.npy' extension. There are various ways by which we can import a text file in a NumPy array.

| Importing & Exporting | Example |
|---|---|
| Saving array on disk | np.save("file", np.arange(5)) |
| Loading a file | np.load("file.npy") |
| Importing a Text File | np.loadtxt('file.txt') |
| Importing CSV File | np.genfromtxt('file.csv',delimiter=',') |
| Write Text File | np.savetxt('file.txt',arr,delimiter=' ') |

### Saving and loading on Disk

Numpy arrays can be stored on the disk using the save() function and loaded using the load() function.

### Python3

```
# the array is saved in the file geekfile.npy
np.save("geekfile", np.arange(5))

# the array is loaded into b
print(np.load("geekfile.npy"))
```

Output:

```
[0 1 2 3 4]
```

### Saving in a Text File

Numpy arrays can be stored on a text file using the savetxt() function.

### Python3

```
x = np.arange(0, 10, 1)
print(x)

# X array saved in geekfile.txt
c = np.savetxt("geekfile.txt", x, delimiter =', ')
```

Output:

```
[0 1 2 3 4 5 6 7 8 9]
```

### Loading from a Text File

The “myfile.txt” has the following content which is loaded using the loadtxt() function.

```
0 1 2
3 4 5
```

## Python3

```
d = np.loadtxt("myfile.txt")
print(d)
```

Output:

```
[[0. 1. 2.]
 [3. 4. 5.]]
```

### Loading a CSV file

We can also load a CSV file in Python using Numpy using another method called genfromtxt() function. The ‘myfilecsv’ has the following content:

```
1,2,3
4,5,6
```

## Python3

```
Data = np.genfromtxt("files\myfile.csv", delimiter=",")
print(Data)
```

Output:

```
[[1. 2. 3.]
 [4. 5. 6.]]
```

## 5. Sorting Array

Numpy arrays can be sorted using the sort() method based on their axis.

| Sorting NumPy Array | Example |
|---|---|
| Sorting 1D Array | arr.sort() |
| Sorting along the first axis of the 2D array | np.sort(a, axis = 0) |

### Example 1: One-Dimensional array

Python3

```
# sorting a one dimensional
# numpy array using numpy.sort()
a = np.array([12, 15, 10, 1])
print("Array before sorting",a)
a.sort()
print("Array after sorting",a)
```

Output:

```
Array before sorting [12 15 10  1]
Array after sorting [ 1 10 12 15]
```

Example 2: Two-Dimensional array

Python3

```
# sorting a two dimensional
# numpy array using numpy.sort()
# sort along the first axis
a = np.array([[12, 15], [10, 1]])
arr1 = np.sort(a, axis = 0)
print ("Along first axis : \n", arr1)

# sort along the last axis
a = np.array([[10, 15], [12, 1]])
arr2 = np.sort(a, axis = -1)
print ("\nAlong Last axis : \n", arr2)

a = np.array([[12, 15], [10, 1]])
arr1 = np.sort(a, axis = None)
print ("\nAlong none axis : \n", arr1)
```

Output:

```
Along first axis :
 [[10  1]
 [12 15]]
Along Last axis :
 [[10 15]
 [ 1 12]]
Along none axis :
 [ 1 10 12 15]
```

## 6. NumPy Array Manipulation

NumPy provides a variety of ways by which we can manipulate NumPy arrays to change their shape or size.

| NumPy Array Manipulation | Example |
|---|---|
| Append at the end of the 1D array | np.append(arr, [7]) |
| Append to 2D array column wise | col = np.arange(5, 11).reshape(1, 6) |

| NumPy Array Manipulation | Example |
|---|---|
| | np.append(arr, col, axis=0) |
| Append to 2D array row-wise | row = np.array([1, 2]).reshape(2, 1)<br>np.append(arr, row, axis=1) |
| Inserting an element at a particular index of a 1D array | np.insert(arr, 1, 9) |
| Inserting an element at a particular index of a 2D array | np.insert(arr, 1, 9, axis = 1) |
| Delete an element from the 1D array | np.delete(arr, object) |
| Delete an element from the 2D array | np.delete(arr, object, axis=1) |
| Reshaping the 1D array to a 2D array | np.reshape(n, m) |
| Reshaping the 2D array to a 1D array | arr.reshape((12)) |
| Resizing array | arr.resize(3, 4) |
| Flattening array | arr.flatten() |
| Transposing array | arr.transpose(1, 0) |

## Appending Elements to Array

Numpy arrays can be manipulated by appending the new values at the end of the array using the append() function

Example 1: One-Dimensional Array

### Python3

```
# Adding the values at the end
# of a numpy array
print("Original Array:", arr)

# appending to the array
arr = np.append(arr, [7])
print("Array after appending:", arr)
```

Output:

```
Original Array: [[ 1.  2.  3.  4.]
 [ 5.  6.  7.  8.]
 [ 9. 10. 11. 12.]]
Array after appending: [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12.  7.]
```

Example 2: N-Dimensional Array

Python3

```
# Adding the values at the end
# of a numpy array
arr = np.arange(1, 13).reshape(2, 6)
print("Original Array")
print(arr, "\n")

# create another array which is
# to be appended column-wise
col = np.arange(5, 11).reshape(1, 6)
arr_col = np.append(arr, col, axis=0)
print("Array after appending the values column wise")
print(arr_col, "\n")

# create an array which is
# to be appended row wise
row = np.array([1, 2]).reshape(2, 1)
arr_row = np.append(arr, row, axis=1)
print("Array after appending the values row wise")
print(arr_row)
```

Output:

```
Original Array
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]
Array after appending the values column wise
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [ 5  6  7  8  9 10]]
Array after appending the values row wise
[[ 1  2  3  4  5  6  1]
 [ 7  8  9 10 11 12  2]]
```

Inserting Elements into the Array

Numpy arrays can be manipulated by inserting them at a particular index using insert() function.

Example 1: One-Dimensional Array

Python3

```
arr = np.asarray([1, 2, 3, 4])
# Python Program illustrating numpy.insert()
print("1D arr:", arr)
print("Shape:", arr.shape)

# Inserting value 9 at index 1
a = np.insert(arr, 1, 9)
```

```
print("\nArray after insertion:", a)
print("Shape:", a.shape)
```

Output:

```
1D arr: [1 2 3 4]
Shape: (4,)
Array after insertion: [1 9 2 3 4]
Shape: (5,)
```

### Removing Elements from Numpy Array

Elements from a NumPy array can be removed using the delete() function.

Example 1: One-Dimensional Array

Python3

```
# Python Program illustrating
# numpy.delete()
print("Original arr:", arr)
print("Shape : ", arr.shape)

# deletion from 1D array
object = 2
a = np.delete(arr, object)
print("\ndeleteing the value at index {} from array:\n {}".format(object,a))
print("Shape : ", a.shape)
```

Output:

```
Original arr: [1 2 3 4]
Shape :  (4,)
deleteing the value at index 2 from array:
 [1 2 4]
Shape :  (3,)
```

### Reshaping Array

NumPy arrays can be reshaped, which means they can be converted from one dimension array to an N-dimension array and vice-versa using the reshape() method. The reshape() function does not change the original array.

Example 1: Reshaping NumPy one-dimension array to a two-dimension array

Python3

```
# creating a numpy array
array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16])

# printing array
print("Array: " + str(array))

# reshaping numpy array
# converting it to 2-D from 1-D array
reshaped1 = array.reshape((4, array.size//4))

# printing reshaped array
```

```
print("First Reshaped Array:")
print(reshaped1)

# creating another reshaped array
reshaped2 = np.reshape(array, (2, 8))

# printing reshaped array
print("\nSecond Reshaped Array:")
print(reshaped2)
```

Output:

```
Array: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
First Reshaped Array:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Second Reshaped Array:
[[ 1  2  3  4  5  6  7  8]
 [ 9 10 11 12 13 14 15 16]]
```

Example 2: Reshaping NumPy from a two dimension array to a one-dimension array.

## Python3

```
# printing array
print(" 2-D Array:")
print(arr)

# reshaping numpy array
# converting it to 1-D from 2-D array
reshaped = arr.reshape((12))

# printing reshaped array
print("Reshaped 1-D Array:")
print(reshaped)
```

Output:

```
 2-D Array:
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]
Reshaped 1-D Array:
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

### Resizing an Array

Numpy arrays can be resized using the resize() function. It returns nothing but changes the original array.

## Python3

```
# Making a random array
arr = np.array([1, 2, 3, 4, 5, 6])
# Required values 12, existing values 6
arr.resize(3, 4)
```

```
print(arr)
```

Output:

```
[[1 2 3 4]
 [5 6 0 0]
 [0 0 0 0]]
```

### Flatten a Two Dimensional array

The flatten() function of Numpy module is used to convert a 2-dimensional array to a 1-dimensional array. It returns a copy of the original array.

## Python3

```
# Two dimensional numpy array
list_1 = [1, 2, 3, 4]
list_2 = [5, 6, 7, 8]
arr = np.array([list_1, list_2])

print(arr.flatten())
```

Output:

```
[1 2 3 4 5 6 7 8]
```

### Transpose

Numpy two-dimensional array can be transposed using the transpose() function.

## Python3

```
# making a 3x3 array
gfg = np.array([[1, 2],
                [4, 5],
                [7, 8]])

# before transpose
print(gfg, end ='\n\n')

# after transpose
print(gfg.transpose(1, 0))
```

Output:

```
[[1 2]
 [4 5]
 [7 8]]
[[1 4 7]
 [2 5 8]]
```

## 7. Combining and Splitting Commands

| Combining and Splitting | Example |
|---|---|
| Combining Arrays | np.concatenate((arr1, arr2), axis = 0) |
| Splitting array | np.split(arr, 3, 1) |
| Horizontal Split | np.hsplit(arr, 3) |
| Vertical Split | np.vsplit(a, 3) |

### Combining Numpy Arrays

Combining two arrays into a single Numpy array can be done using concatenate() method.

## Python3

```
arr1 = np.array([[2, 4], [6, 8]])
arr2 = np.array([[3, 5], [7, 9]])

# combining on axis 0
gfg = np.concatenate((arr1, arr2), axis = 0)
print(gfg)

# combining on axis 1
gfg = np.concatenate((arr1, arr2), axis = 1)
print("\n", gfg)

# combining on axis None
gfg = np.concatenate((arr1, arr2), axis = None)
print("\n", gfg)

# stacking two arrays on one another
print(np.stack((arr1, arr2), axis=1))
```

Output:

```
 [[2 4]
  [6 8]
  [3 5]
  [7 9]]
  [[2 4 3 5]
  [6 8 7 9]]
  [2 4 6 8 3 5 7 9]
 [[[2 4]
   [3 5]]
  [[6 8]
   [7 9]]]
```

### Splitting Numpy Arrays

Example 1: using split()

Numpy arrays can be split into multiple arrays horizontally or vertically.

## Python3

```
# Making of  a 3x3 array
a = np.arange(9).reshape(3, 3)
print(a)

# Horizontal splitting of array 'a'
# using 'split' with axis parameter = 1.
print("Splitted array in horizontal form:\n", np.split(a, 3, 1))

# Vertical splitting of array 'a'
# using 'split' with axis parameter = 0.
print("\nSplitted array in vertical form:\n", np.split(a, 3, 0))
```

Output:

```
 [[0 1 2]
  [3 4 5]
  [6 7 8]]
 Splitted array in horizontal form:
  [array([[0],
        [3],
        [6]]), array([[1],
        [4],
        [7]]), array([[2],
        [5],
        [8]])]
 Splitted array in vertical form:
  [array([[0, 1, 2]]), array([[3, 4, 5]]), array([[6, 7, 8]])]
```

Example 2: using hsplit()

The hsplit() splits the Numpy array in multiple horizontal arrays.

## Python3

```
# Horizontal splitting of array
# 'a' using np.hsplit().
print(a)
print("Splitted array in horizontally to form:", np.hsplit(a, 3))
```

Output:

```
 [[0 1 2]
  [3 4 5]
  [6 7 8]]
 Splitted array in horizontally to form: [array([[0],
        [3],
        [6]]), array([[1],
        [4],
        [7]]), array([[2],
        [5],
        [8]])]
```

Example 3: using vsplit()

The vsplit() splits the Numpy array into multiple vertical arrays.

Python3

```
# Vertical splitting of array 'a'
# using np.vsplit().
print("Splitted array in vertically to form:", np.vsplit(a, 3))
```

Output:

```
Splitted array in vertically to form:
 [array([[0, 1, 2]]), array([[3, 4, 5]]), array([[6, 7, 8]])]
```

## 8. Indexing, Slicing and Subsetting

Different ways of Indexing the Numpy array are as follows:

| Indexing, Slicing and Subsetting | Example |
|---|---|
| Subsetting | arr[np.array([1, 3, -3])] |
| Sclicing | arr[-2:7:1] |
| Integer Indexing | a[[0 ,1 ,2 ],[0 ,0 ,1]] |
| Boolean Indexing | a[a>50] |

Subsetting Numpy Array

Python3

```
# Index values can be negative.
print(arr)
print("Elements are:", arr[np.array([1, 3, -3])])
```

Output:

```
[1 2 3 4 7]
Elements are: [2 4 3]
```

Slicing Numpy Array

The ":" operator means all elements till the end.

Python3

```
print(arr)
```

```
# a[start:stop:step]
print("a[-2:7:1] = ",arr[-2:7:1])

print("a[1:] = ",arr[1:])
```

Output:

```
[1 2 3 4 7]
a[-2:7:1] =  [4 7]
a[1:] =  [2 3 4 7]
```

### Indexing Numpy Array

Numpy array indexing is of two types: Integer indexing and Boolean indexing.

## Python3

```
# Integer Indexing
a = np.array([[1 ,2 ],[3 ,4 ],[5 ,6 ]])
print(a[[0 ,1 ,2 ],[0 ,0 ,1]])

# Boolean Indexing
a = np.array([10, 40, 80, 50, 100])
print(a[a>50])
```

Output:

```
[1 3 6]
[ 80 100]
```

## 9. Copying and Viewing Array

NumPy arrays can be manipulated with and without making a copy of an array object. When an array is copied with a normal assignment, it uses the exact same id as the original array. Whereas when a deep copy of the array object is made, a completely new array is created with a different id. This does not affect the original array when any changes are made to the newly copied array.

| Copying and Viewing Array | Example |
|---|---|
| Coping to new memory space | arr.copy() |
| Shallow Copy | arr.view() |

### Copying Array

Let us see different ways of copying and viewing numpy arrays.

Shallow copy

## Python3

```
# Copying Numpy array with normal assignment
nc = arr
```

```
# both arr and nc have same id
print("Normal Assignment copy")
print("id of arr:", id(arr))
print("id of nc:", id(nc))

# updating nc
nc[0]= 12

# printing the values
print("original array:", arr)
print("assigned array:", nc)
```

Output:

```
Normal Assignment copy
id of arr: 139656514595568
id of nc: 139656514595568
original array: [12  2  3  4]
assigned array: [12  2  3  4]
```

Deep Copy

Python3

```
# Creating a different copy of NumPy
# array creating copy of array
c = arr.copy()

# both arr and c have different id
print("id of arr:", id(arr))
print("id of c:", id(c))

# changing original array
# this will not effect copy
arr[0] = 12

# printing array and copy
print("original array:", arr)
print("copy:", c)
```

Output:

```
id of arr: 139656514596912
id of c: 139656514596432
original array: [12  2  3  4]
copy: [1 2 3 4]
```

Viewing Array

The view is also known as a shallow copy which is just a view of the original array. It has a separate id but any changes made to the original will also reflect on the view.

Python3

```
# Creating a view of a
# NumPy array
# creating view
v = arr.view()
```

```
# both arr and v have different id
print("id of arr:", id(arr))
print("id of v:", id(v))

# changing original array
# will effect view
arr[0] = 12

# printing array and view
print("original array:", arr)
print("view:", v)
```

Output:

```
id of arr: 139656514598640
id of v: 139656514599216
original array: [12  2  3  4]
view: [12  2  3  4]
```

## 10. NumPy Array Mathematics

### Arithmetic Operations

Numpy arrays can perform arithmetic operations like addition, subtraction, multiplication, division, mod, remainder, and power.

| Arithmetic Operations | Example |
|---|---|
| Adds elements of 2 Array | np.add(a, b) |
| Substracts elements of 2 Array | np.subtract(a, b) |
| Multiply elements of 2 Array | np.multiply(a, b) |
| Divide elements of 2 Array | np.divide(a, b) |
| Modulo of elements of 2 Array | np.mod(a, b) |
| Remainder of elements of 2 Array | np.remainder(a,b) |
| Power | np.power(a, b) |

| Arithmetic Operations | Example |
|---|---|
| of elements of 2 Array | |
| Exponant<br>value of elements of 2 Array | np.exp(b) |

## Python3

```
# Defining both the matrices
a = np.array([5, 72, 13, 100])
b = np.array([2, 5, 10, 30])

# Performing addition using numpy function
print("Addition:", np.add(a, b))

# Performing subtraction using numpy function
print("Subtraction:", np.subtract(a, b))

# Performing multiplication using numpy function
print("Multiplication:", np.multiply(a, b))

# Performing division using numpy functions
print("Division:", np.divide(a, b))

# Performing mod on two matrices
print("Mod:", np.mod(a, b))

#Performing remainder on two matrices
print("Remainder:", np.remainder(a,b))

# Performing power of two matrices
print("Power:", np.power(a, b))

# Performing Exponentiation
print("Exponentiation:", np.exp(b))
```

### Output:

```
Addition: [  7  77  23 130]
Subtraction: [ 3 67  3 70]
Multiplication: [  10  360  130 3000]
Division: [ 2.5        14.4         1.3         3.33333333]
Mod: [ 1  2  3 10]
Remainder: [ 1  2  3 10]
Power: [                25          1934917632        137858491849
 1152921504606846976]
Exponentiation: [7.38905610e+00 1.48413159e+02 2.20264658e+04 1.06864746e+13]
```

### Comparison

Numpy array elements can be compared with another array using the array_equal() function.

## Python3

```
an_array = np.array([[1, 2], [3, 4]])
```

```
another_array = np.array([[1, 2], [3, 4]])

comparison = an_array == another_array
equal_arrays = comparison.all()

print(equal_arrays)
```

Output:

```
True
```

### Vector Math

Numpy arrays can also determine square root, log, absolute, sine, ceil, floor, and round values.

| Vector Math | Example |
|---|---|
| Square root of each element | np.sqrt(arr) |
| Log value of each element | np.log(arr) |
| Absolute value of each element | np.absolute(arr) |
| Sine value of each element | np.sin(arr) |
| Ceil value of each element | np.ceil(arr) |
| Floor value of each element | np.floor(arr) |
| Round value of each element | np.round_(arr) |

## Python3

```
arr = np.array([.5, 1.5, 2.5, 3.5, 4.5, 10.1])

# applying sqrt() method
print("Square-root:", np.sqrt(arr))

# applying log() method
```

```
print("Log Value: ", np.log(arr))

# applying absolute() method
print("Absolute Value:", np.absolute(arr))

# applying sin() method
print("Sine values:", np.sin(arr))

# applying ceil() method
print("Ceil values:", np.ceil(arr))

# applying floor() method
print("Floor Values:", np.floor(arr))

# applying round_() method
print ("Rounded values:", np.round_(arr))
```

Output:

```
Square-root: [0.70710678 1.22474487 1.58113883 1.87082869 2.12132034
3.17804972]
Log Value:  [-0.69314718  0.40546511  0.91629073  1.25276297  1.5040774
2.31253542]
Absolute Value: [ 0.5  1.5  2.5  3.5  4.5 10.1]
Sine values: [ 0.47942554  0.99749499  0.59847214 -0.35078323 -0.97753012
-0.62507065]
Ceil values: [ 1.  2.  3.  4.  5. 11.]
Floor Values: [ 0.  1.  2.  3.  4. 10.]
Rounded values: [ 0.  2.  2.  4.  4. 10.]
```

## Statistic

Numpy arrays also perform statistical functions such as mean, summation, minimum, maximum, standard deviation, var, and correlation coefficient.

| Statistic | Example |
|---|---|
| Mean | np.mean(arr) |
| Median | np.median(arr) |
| Summation | np.sum(arr, dtype = np.uint8) |
| Maximum | np.max(arr) |
| Minimum value | np.min(arr) |
| Variance | np.var(arr, dtype = np.float32) |
| Standard Deviation | np.std(arr, dtype = np.float32) |

| Statistic | Example |
| --- | --- |
| Correlation Coefficient | np.corrcoef(array1, array2) |

## Python3

```
# 1D array
arr = [20, 2, 7, 1, 34]

# mean
print("mean of arr:", np.mean(arr))

# median
print("median of arr:", np.median(arr))

# sum
print("Sum of arr(uint8):", np.sum(arr, dtype = np.uint8))
print("Sum of arr(float32):", np.sum(arr, dtype = np.float32))

# min and max
print("maximum element:", np.max(arr))
print("minimum element:", np.min(arr))

# var
print("var of arr:", np.var(arr))
print("var of arr(float32):", np.var(arr, dtype = np.float32))

# standard deviation
print("std of arr:", np.std(arr))
print ("More precision with float32", np.std(arr, dtype = np.float32))
```

### Output:

```
mean of arr: 12.8
median of arr: 7.0
Sum of arr(uint8): 64
Sum of arr(float32): 64.0
maximum element: 34
minimum element: 1
var of arr: 158.16
var of arr(float32): 158.16
std of arr: 12.576167937809991
More precision with float32 12.576168
```

### corrcoef

## Python3

```
# create numpy 1d-array
array1 = np.array([0, 1, 2])
array2 = np.array([3, 4, 5])

# pearson product-moment correlation
# coefficients of the arrays
rslt = np.corrcoef(array1, array2)
```

```
print(rslt)
```

Output:

```
[[1. 1.]
 [1. 1.]]
```

## Benefits of Using NumPy Cheat Sheet

NumPy Cheat Sheet comes with advantages that make it essential for Python programmers and data scientists. Here are some of the key benefits of NumPy:

1. Efficient Data Handling: NumPy provides a robust framework for handling large datasets efficiently, enabling faster data processing and manipulation.
2. Mathematical Functions: The library includes an extensive collection of mathematical functions for operations like trigonometry, statistics, linear algebra, and more.
3. Broadcasting: NumPy allows broadcasting, which enables element-wise operations on arrays of different shapes, reducing the need for explicit loops.
4. Interoperability: It integrates with other libraries like Pandas, SciPy, and Matplotlib, improving its functionality for data analysis and visualization.
5. Memory Optimization: NumPy optimizes memory usage, making it ideal for working with large datasets without consuming excessive RAM.
6. Multidimensional Arrays: The library supports multidimensional arrays, enabling easy manipulation of complex data structures.
7. Open-source and Community Support: NumPy is open-source, and its active community provides regular updates, bug fixes, and additional functionalities.

## Applications of NumPy

The various applications of Numpy other than data analysis are given below

- Scientific Computing
- Data Analysis and Preprocessing
- Image Processing
- Machine Learning
- Signal Processing

## Feature of NumPy

Here are some features of Numpy that why Numpy is famous for data analysis and scientific computing

- It is a powerful N-dimensional array object “ndarray“.
- Numpy offers a wide range of collections of Mathematical Functions.
- It easily Integrates with low-level languages such as C/C++ and Fortran Also.
- It offers a comprehensive range of broadcasting functions for dealing with arrays of different dimensions.

## Conclusion

In Conclusion, NumPy arrays can be used for math operations, data loading and storing, and array indexing. We covered all array manipulation, import/export, and statistical techniques that are crucial. Apart from that, this Numpy Cheat Sheet is thoughtfully organized and categorized, making it easy for developers to quickly find the functions they need for specific

use cases. Whether it's sorting and filtering array data, or creating and manipulating an array it covers at all. By utilizing this resource, data analysts can enhance their productivity and efficiency in working with Numpy, ultimately leading to smoother and more successful data analysis projects.

> *Don't miss our Python cheat sheet for data science, covering important libraries like Scikit-Learn, Bokeh, Pandas, and Python basics.*

## NumPy Cheat Sheet – FAQs

### 1. What is NumPy Cheat Sheet?

> *When your memory fails or you prefer not to rely on "Python help()" in the command line, this NumPy cheat sheet comes to the rescue. It is hard to memorize all the important NumPy funtions by heart, so print this out or save it to your desktop to resort to when you get stuck.*

### 2. What is the Full Form of Numpy?

> *NumPy is a Combination of Two words Numerical and Python. It made it because this Python library deal with all the numerical operations on an array.*

### 3. How is data stored in NumPy?

> *NumPy arrays consist of two major components: the underlying data buffer (referred to as the data buffer from this point forward) and the metadata associated with the data buffer. The data buffer, reminiscent of arrays in C or Fortran, represents a contiguous and unvarying memory block comprising fixed-sized data elements.*

### 4. What is the seed function in NumPy?

> *The seed() function in NumPy is used to set the random seed of the NumPy pseudo-random number generator. It offers a crucial input that NumPy needs to produce pseudo-random integers for random processes and other applications. By default, the random number generator uses the current system time.*

Ready to dive into the future? Mastering Generative AI and ChatGPT is your gateway to the cutting-edge world of AI. Perfect for tech enthusiasts, this course will teach you how to

leverage Generative AI and ChatGPT with hands-on, practical lessons. Transform your skills and create innovative AI applications that stand out. Don't miss out on becoming an AI expert – Enroll now and start shaping the future!

T tanya... + Follow 9

Previous Article Next Article

## Similar Reads

### SQL Cheat Sheet ( Basic to Advanced)

In this article, we will explore the ultimate SQL cheat sheet with the PDF download, covering a basic to advance of SQL commands, Joins in SQL, CRUD Operations, SQL...

15 min read

### jQuery Cheat Sheet – A Basic Guide to jQuery

What is jQuery?jQuery is an open-source, feature-rich JavaScript library, designed to simplify the HTML document traversal and manipulation, event handling, animation,...

15+ min read

### Complexity Cheat Sheet for Python Operations

Python built-in data structures like list, sets, dictionaries provide a large number of operations making it easier to write concise code but not being aware of their...

2 min read

### Tkinter Cheat Sheet

Tkinter, the standard GUI library for Python, empowers developers to effortlessly create visually appealing and interactive desktop applications. This cheat sheet offer...

8 min read

### CSS Cheat Sheet - A Basic Guide to CSS

What is CSS? CSS i.e. Cascading Style Sheets is a stylesheet language used to describe the presentation of a document written in a markup language such as HTM...

13 min read

View More Articles

**Article Tags :** Python Cheat Sheet GFG Sheets Python-numpy

**Practice Tags :** python

GeeksforGeeks
Sanchhaya Education Private Limited
Corporate & Communications Address:- A-143, 9th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305) | Registered Address:- K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305

GET IT ON Google Play
Download on the App Store

Company
About Us
Legal
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
GeeksforGeeks Community

Languages
Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial
Tutorials Archive

DSA
Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

Data Science & ML
Data Science With Python
Data Science For Beginner
Machine Learning Tutorial
ML Maths
Data Visualisation Tutorial
Pandas Tutorial
NumPy Tutorial
NLP Tutorial
Deep Learning Tutorial

Web Technologies
HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Python Tutorial
Python Programming Examples
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question
Django

Computer Science
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths

DevOps
Git
Linux
AWS
Docker
Kubernetes
Azure

Software Development
Software Testing

## System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

## School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

GCP
DevOps Roadmap

## Inteview Preparation

Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation
Puzzles

## GeeksforGeeks Videos

DSA
Python
Java
C++
Web Development
Data Science
CS Subjects

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved