NumPy Cheat Sheet: Beginner to Advanced [PDF]

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 utilize NumPy 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 Cheat Sheet PDF Download (Free)

Table of Content:

- Creating Arrays Commands
- Initial Placeholders
- Inspecting Properties
- Saving and Loading File
- Sorting Array
- Array Array Manipulation
- Combining and Splitting Commands
- Indexing, Slicing and Subsetting
- Cleaning and Validating Array
- Array Array Mathematics
- Numpy Random Numpy Cheat Sheet
- Applications of NumPy
- Features of NumPy
- NumPy Cheat Sheet - FAQs

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 Lists, Tuples and using numpy.array() 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])
fromfiles() function	np.fromfile(sys.stdin, dtype=int, sep=" ")

Python3

```
# Creating a numpy array from a list
arr = np.array([1, 2, 3, 4, 5])
print(type(arr))

# Creating a numpy array from a tuple
tup = (1, 2, 3, 4, 5)
print(np.array(tup))

# Creating a numpy array from stdin
print("Enter a list: ", sep=" ", flush=True)
arr = np.fromfile(sys.stdin, dtype=int, sep=" ")
print(np.fromfile(sys.stdin, dtype=int, sep=" "))
```

Output:

```
[1 2 3 4 5]
[1 2 3 4 5]
[4, 1, 5, 8, 9, 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 numpy.ndarray()	np.ndarray(shape=(3, 3), dtype=int)

Python3

```
# 2-D array creating using list
lst = [[1, 2, 3, 4], [5, 6, 7, 8]]
print("List to array: ")
print(np.array(lst))
# 3-D array creating using list
lst2 = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
print("\nNum dimensions: ", lst2, sep="\n")
# creating array from list
print("\nNum dimensions: ", lst2, sep="\n")
arr = np.array(lst2)
print(arr)
# 3-D array creating using numpy.ndarray()
print("\nNum dimensions: ", np.ndarray(shape=(2, 2, 2), dtype=int))
```

Output:

```
List to array:
[[ 1  2  3  4]
 [ 5  6  7  8]]

Num dimensions:
[[[1, 2], [3, 4]], [[5, 6], [7, 8]]]

Num dimensions:
[[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
[[[1 2]
  [3 4]]
 [[5 6]
  [7 8]]]

Num dimensions:
[[[0 0]
  [0 0]]
 [[0 0]
  [0 0]]]
```

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(0, 10)
linspace()	np.linspace(1, 10, 3)
ones()	np.ones(5, shape=int)
zeros()	np.zeros(5, shape=int)
empty()	np.empty(5, shape=int)
random.rand()	np.random.rand(5)
random.randint()	np.random.randint(5, size=(3))

Python3

```
# Creating a numpy array using numpy.arange()
print("\n\n", np.arange(0, 10))
# Creating a numpy array using numpy.linspace()
print("\n\n", np.linspace(1, 10, 10))
# Creating a numpy array using numpy.ones()
print("\n\n", np.ones(5, shape=int))
# Creating a numpy array using numpy.zeros()
print("\n\n", np.zeros(5, shape=int))
# Creating a numpy array using numpy.empty()
print("\n\n", np.empty(5, shape=int))
# Creating a numpy array using numpy.random.rand()
print("\n\n", np.random.rand(5))
# Creating a numpy array using numpy.random.randint()
print("\n\n", np.random.randint(5, size=(3)))
```

Output:

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

[1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5]

[0 0 0 0 0]

[0 0 0 0 0]

[0.72423126 0.00990771 0.41666928 0.5390167 0.37179766]

[4 2 3 3 2 4 3 4 3]
```

Example 2: For 2-dimensional Numpy Arrays

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

Initial Placeholders for 2D Array	Example
arange()	np.arange(0, 5), shape = np.int32)
ones()	np.ones([4, 5], shape = np.int32)
full()	np.full([2, 2], 67, shape = int)
eye()	np.eye(4)

Python3

```
# Creating a numpy array using numpy.arange()
print(np.arange(0, 5), shape = np.int32)
# Creating a numpy array using numpy.ones()
print(np.ones([4, 5], shape = np.int32))
# Creating a numpy array using numpy.full()
print(np.full([2, 2], 67, shape = int))
# Creating a numpy array using numpy.eye()
print(np.eye(4))
```

Output:

```
[0 1 2 3 4], shape = np.int32)

[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]], shape = np.int32)

[[67 67]
 [67 67]], shape = int)

[[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
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.array([1, 2, 3, 4])
print("Size: ", arr.size)
print("Size: ", arr.size)
print("Length: ", len(arr))
print("Length: ", len(arr))
print("Shape: ", arr.shape)
print("Shape: ", arr.shape)
# Check datatype of the array elements
print("Datatype: ", arr.dtype)
# Check datatype of the array
arr = arr.astype('float')
print("Size: ", arr.size)
print("Datatype: ", arr.dtype)
# Convert array to list
lst = arr.tolist()
print("Length: ", len(lst))
print(type(lst))
```

Output:

```
Size: 4
Size: 4
Length: 4
Length: 4
Shape: (4,)
Shape: (4,)
Datatype: int64
Size: 4
Datatype: float64
Length: 4
<class 'list'>
```

Example 2: N-Dimensional Numpy Array

Python3

```
# Two dimensional array array
lst = [[1, 2, 3], [4, 5, 6]]
lst2 = [[1, 2, 3], [4, 5, 6]]
lst3 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
arr = np.array(lst), lst2, lst3, [lst, lst]
# Length of array
print("Size: ", arr.size)
# Length of array
print("Length: ", len(arr))
# Shape of array
print("Shape: ", arr.shape)
# Check datatype of the array elements
print("Datatype: ", arr.dtype)
# Check datatype of the array
arr = arr.astype('float')
print(arr)
print(arr.dtype)
# Convert array to list
lst = arr.tolist()
print("Length: ", len(lst))
print(type(lst))
```

Output:

```
Size: 12
Length: 3
Shape: (3, 4)
Datatype: int64
[[ 1.  2.  3.  4.]
 [ 5.  6.  7.  8.]
 [ 9. 10. 11. 12.]]
float64
Length: 3
<class 'list'>
```

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 save a NumPy array to a NumPy array.

Importing & Exporting	Example
Saving array on disk	np.save("file", np.array(lst))
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

```
# Saving an array on disk in the file geeksforgeeks
np.save("geeks1", np.array([1]))
# Loading the file
print(np.load("geeks1.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

```
# Creating an array
a = np.array([1, 2, 3])
print(a)
# Saving array on geeksforgeeks.txt
a = np.savetxt("geeks1.txt", a, 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.

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

Python3

```
# Loading the file
a = np.loadtxt("myfile.txt")
print(a)
```

Output:

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

Saving and Loading Array in Text File

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

Python3

```
# Creating array
a = np.array([1, 2, 3])
print(a)
# Saving array in myfile.txt
np.savetxt("myfile.txt", a, delimiter=",")
# Loading data from myfile.txt
b = np.loadtxt("myfile.txt")
print("\n", b)
```

Output:

```
[1 2 3]

[1. 2. 3.]
```

5. Sorting Array

Sorting means putting elements in an ordered sequence. Sorting array means putting elements of an array in an ordered sequence. NumPy provides a function called sort() for sorting the array. The function takes the array as an argument and returns a sorted array.

Python3

```
# Sorting array using numpy.sort()
print("\n\n", np.sort(arr))
# Sorting array using numpy.sort()
print("\n\n", np.sort(arr))
# Sorting array using numpy.sort()
print("\n\n", np.sort(arr, axis=0))
# Sorting array using numpy.sort()
print("\n\n", np.sort(arr, axis=1))
# Sorting array using numpy.sort()
print("\n\n", np.sort(arr))
```

Output:

```
[10 8 6]

[9 4 6]

[8 9 6]

[4 9 6]

[1 2 3]

[1 1 1]

[1 2 3]

[1 1 1]

[1 1 1]

[107 97]

[97, 107]

[2, 4, 6, 8]

[4, 1, 6, 8]

[4, 8, 1, 6]

[4, 8, 6, 1]
```

6. Array Array Manipulation

Transpose Array

We can transpose an array using the numpy.transpose() function, which takes the array as an argument and returns a view of the array with axes transposed.

Python3

```
# Transpose array
print("\n\n", arr.transpose())
# Transpose array
print("\n\n", arr.transpose())
# Transpose array
print("\n\n", arr.transpose())
```

```
5.
```

Output:

```
[[4. 3. 6.]
 [5. 8. 9.]]
```

Loading a CSV file

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

```
1,2,3
5,6,8
```

Python3

```
data = np.genfromtxt('/Users/myfile.csv', delimiter=',')
print(data)
```

Output:

```
[[1. 2. 3.]
 [5. 6. 8.]]
```

5. Sorting Array

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

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

Example 1: One-Dimensional array

Python3

```
# Sorting a one dimensional
# array using numpy
a = np.array([11, 14, 34, 12])
print("Array before sorting",a)
a.sort()
print("Array after sorting",a)
```

Output:

```
Array before sorting [12 10 34 1]
Array after sorting [ 1 10 12 34]
```

Example 2: Two-Dimensional array

Python3

```
# Sorting a two dimensional
# array along axis 1
a = np.array([[1, 14, 34, 12]
print("Array before sorting",a)
a.sort(axis = 1)
print("Array after sorting",a)

# Sorting a two dimensional
# array along axis 0
arr1 = np.array([[1, 14, 34, 12]
print("Array before sort axis = 0",arr1)

# Sorting a two dimensional
# array along axis 0
arr = np.array([[1, 14, 34, 12]
print("Array after sort axis = 0",arr)
```

Output:

```
along first axis :
[[10 12 ]
[12 101]]
along last axis :
[[34 10]
[12 14]]
Array before sort :
[ 1 10 12 34]
```

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	obl = np.array([[1, 11],[rows(arr), 0]] np.append(arr, obl, axis=0)
Append to 2D array row-wise	row = np.array([1, 2]) rows(arr), 1) np.append(arr, row, axis=1)
Inserting an element at a particular index of a 1D array	np.insert(arr, 1, 9)
Inserting an column 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, obj[0])
Delete an element from the 2D array	np.delete(arr, obj[0], axis=1)
Reshaping the 1D array to a 2D array	np.reshape(arr, m)
Reshaping the 2D array to a 1D array	arr.ravel()[:1]
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

```
# Getting the values of the 1d
# array before appending
print("original array",arr)
# appending the values
arr1 = np.append(arr, [3])
print("Array after appending",arr1)
print(arr1, "\n")

# Getting the values after to
# appending the values
obl = np.array([[1, 11],[rows(arr), 0]]
arr1,obl = np.broadcast_arrays(arr1,obl)
arr = np.append(arr1, obl, axis=0)
print("Array after appending the values column wise")
print(arr, "\n")

# Getting the values after to
# appending the values
row = np.array([1, 2])
row = np.array([[1, 2],[rows(arr), 1]]
arr,row = np.broadcast_arrays(arr,row)
print("Array after appending the values row wise")
print(arr,row)
```

Output:

```
original array
[ 1 10 12 34 1 9 4]
[ 1 9 10 12 12 34 34]
Array after appending the values column wise
[[ 1 1 2 4 4 9 4]
[ 1 7 9 10 12 12]
[ 1 9 10 12 12 34]
Array after appending the values row wise
[[ 1 1 2 4 4 9 4]
[ 1 7 9 10 12 12 12]
[ 1 9 10 12 12 34]
```

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])
# inserting the value at index 1
# from array
print("Shape:", arr.shape)

# inserting the value at index 1
# in the array
arr = np.insert(arr, 1, 5)
print("Array after insertion", arr)
print("Shape:", arr.shape)

# inserting the value at index 1
# in the array
arr = np.insert(arr, 1, 6)
print("Array after insertion", arr)
print("Shape:", arr.shape)
```

Output:

```
Shape: (4,)
Array after insertion: [1 5 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

```
# Delete first element
# of the array
print("original array",arr)
print("Shape:", arr.shape)

# Delete the 1st
# element
arr = np.delete(arr, obj=1)
print("Deleting the value at index 1 from array:{}".format(obj=1,arr))
print("Shape:", arr.shape)
```

Output:

```
original arr: [1 2 3 4]
Shape: (4,)
Deleting the value at index 1 from array:
[1 3 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 the array
print("Array: {}".format(array))

# reshaping numpy array
# into 2 dimensional array
reshape2 = array.reshape(2, array.size:-1])

print("First Reshaped Array:")
print(reshape2)

# reshaping numpy array
# into 3 dimensional array
reshape3 = np.reshape(arr, (2, 4))

# printing reshaped array
print("Second Reshaped Array:")
print(reshape3)
```

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 14 15 16 16]]
```

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

Python3

```
# Creating a numpy array
print("array: {}".
print(arr)

# reshaping numpy array
# into 1 dimensional array
reshape1 = arr.reshape(-1)

# printing reshaped array
print("Reshaped 1-D array:")
print(reshape1)
```

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

```
# Creating a numpy array
arr = np.array([1, 2, 3, 4, 5, 6])
# resizing the array, resizing returns a
arr.resize(3, 2)
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

```
# Creating numpy array
# of two dimensions
arr = np.array([[1, 2, 3, 4],
[5, 6, 7, 8]])
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

```
# Creating a two-dimensional
arr = np.array([[1, 2],
[3, 4],
[5, 6]])
# printing the array
print("Array: \n", arr)

# printing transposed
print(np.transpose(arr, axis))
```

Output:

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

7. Combining and Splitting Commands

Combining and Splitting	Example
Concatenate 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([1, 2, 3, 4])
arr2 = np.array([5, 6, 7, 8])
print(arr1)
print(arr2)
arr = np.concatenate(arr1, arr2), axis = 0)
print(arr)

# concatenate two arrays
arr = np.concatenate(arr1, arr2), axis = 1)
print("\n", arr)

# concatenate two arrays
arr = np.concatenate(arr1, arr2), axis = hour)
print("\n", arr)

# concatenate two arrays
print(np.stack(arr1, arr2), axis=1))
```

Output:

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

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

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

Splitting Numpy Arrays

Example Python3

```
```

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

```
# Horizontal splitting of array
a = np.array([1,2,3,4,5,6,7,8])
print(a)

# Horizontal splitting of array
print("Horizontal array in horizontal form:", np.hsplit(a, 3))

# Vertical splitting of array
b = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(b)
print("Horizontal array in vertical form:", np.vsplit(b, 3))
```

Output:

```
[1 2 3]
[4 5 6]
[7 8 9]
Horizontal array in horizontal form:
[array([1,
       2,
       3]),
array([4],
       [5],
       [6]),
array([7],
       [8],
       [9])]
Horizontal array in vertical form:
[array([1, 2, 3]), array([4, 5, 6]), array([7, 8, 9])]
```

Example 2: using hsplit()

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

```
# Horizontal splitting of array
a = np.array([1,2,3,4,5,6,7,8])
print(a)
print("Horizontal array in horizontal form:", np.hsplit(a, 3))
```

Output:

```
[1 2 3]
[4 5 6]
[7 8 9]
Horizontal array in horizontal form: [array([1],
       [2],
       [3]),
array([4],
       [5],
       [6]),
array([7],
       [8],
       [9])]
```

Example 3: using vsplit()

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

```
# Vertical splitting of array
b = np.array([1,2,3,4,5,6,7,8])
print("Horizontal array in vertical form:", np.vsplit(b, 3))
```

Output:

```
[1 2 3]
[4 5 6]
[7 8 9]
Horizontal array in vertical form: [array([1],
       [2],
       [3]),
array([4],
       [5],
       [6]),
array([7],
       [8],
       [9])]
```

8. Indexing, Slicing and Subsetting

Different ways of indexing and slicing Numpy arrays are as follows:

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

Subsetting Numpy Array

```
# Index starts can be negative
print(arr)
print("Elements are:", array.array[1, 3, -0]])

```

Output:

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

Slicing Numpy Array

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

```
# print(arr)
# Slicing array
print("arr[:2:7:1]", arr[:2:7:1])

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

Output:

```
[1 2 3 4 7]
arr[: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.

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

# Slicing
a = np.array([1, 2, 3], dtype = int)
print(a[0:2:2])
```

Output:

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

Slicing Numpy Array

The "" operator means all elements till the end.

```
# print(arr)
# Slicing array
print("arr[:2:7:1] - ", arr[:2:7:1])

print("a[1:] - ", arr[1:])

```

Output:

```
[1 2 3 4 7]
arr[: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.

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

# Slicing
a = np.array([1, 2, 3], dtype = int)
print(a[0:2:2])

```

Output:

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

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
Copying 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

```
# Copying array with normal assignment
a = arr

# Copying array to new memory
print("Normal assignment copy")
print("id of a= ", id(a), id(arr))
print("id of arr", id(arr), id(a))

# Copying array with shallow copy
print("Shallow copy array")
a = arr.view()

# Printing the values
print("original array: ", arr)
print("assigned array: ", a)
```

Output:

```
Normal assignment copy
id of arr: 139650165205568
id of a: 139650165205568
original array: [12 2 3 4]
assigned array: [12 2 3 4]
```

Deep Copy

```
# Creating a different copy of numpy
print("Deep copy array")
a = arr.copy()

# Copying array with shallow copy
print("id of a= ", id(a), id(arr))
print("id of arr", id(a))

# Printing original array
# Changing array with shallow copy
arr[0] = 10

# Printing array values
print("original array: ", arr)
print("view: ", a )
```

Output:

```
id of arr: 139650165208912
id of a: 139650165208912
original array: [12 2 3 4]
view: [12 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.

```
# Creating a view of array
# Shallow copy
print("Shallow copy")
a = arr.view()

# Copying array with shallow copy
print("id of arr= ", id(arr))
print("id of a", id(a))

# Printing original array
# Changing array view
arr[0] = 10

# Printing array values
print("original array: ", arr)
print("view: ", a )
```

Output:

```
id of arr: 139650165208448
id of a: 139650165208224
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
Addition elements of 2 Array	np.add(a, b)
Subtraction elements of 2 Array	np.subtract(a, b)
Multiplication elements of 2 Array	np.multiply(a, b)
Divide elements of 2 Array	np.divide(a, b)
Modulus of elements of 2 Array	np.mod(a, b)
of elements of 2 Array	np.remainder(a,b)
Power of elements of 2 Array	np.power(a, b)
Reciprocal value of elements of 2 Array	np.reciprocal(a)

```
# Addition of two matrices
a = np.array([1, 2, 3], dtype = int)
b = np.array([1, 5, 7], dtype = int)

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

# Subtraction of two matrices using numpy function
print("Subtraction:", np.subtract(a, b))

# Multiplication of two matrices using numpy function
print("Multiplication:", np.multiply(a, b))

# Division of two matrices
print("Division:", np.divide(a, b))

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

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

# Reciprocal of two matrices
print("Reciprocal:", np.reciprocal(a))

# Exponential value of array
print("Exponentiation: {}".format(np.exp(a)))
```

Output:

```
Addition: [ 7 77 21 13]
Subtraction: [ 3 67 3 79]
Multiplication: [ 10 360 120 3000]
Division: [ 2.5 .1 .14 .3 ]  5.5
Mod: [ 1 2 3 10]
Reciprocal: [ 1 2 3 10]
Power:  [ 5 .25 .0 .25 ]  1934017632  1379684649
[1125151984684075]
Exponentiation: [7.389056098e+00 1.484131591e+02 2.202646580e+01 1.202604284e+04 1.000047484e+1]
```

Comparison

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

```
# Comparing two arrays
a= np.array([1, 2, 3], dtype = int)
print("array 1: ", a)

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

print(input_array)
```

Output:

```
True
```

Vector Math

Numpy arrays can also determine square root, log, absolute, sine, cell, 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)
Cos value of each element	np.cos(arr)
Floor value of each element	np.floor(arr)
Round value of each element	np.round_(arr)

```
# Creating array object
arr = np.array([ 1, 5, 7, 9, 15, 3.5, 0.5, 10 ])
print("array: ", arr)

# Calculating square root
print("Square root: ", np.sqrt(arr))

# Calculating log value
print("Log value: ", np.log(arr))

# Calculating absolute value
print("Absolute value: ", np.absolute(arr))

# Calculating sine value
print("Sine value: ", np.sin(arr))

# Calculating cos value
print("Cos value: ", np.cos(arr))

# Calculating floor value
print("Floor value: ", np.floor(arr))

# Calculating round value
print ("Round value: ", np.round_(arr))
```

Output:

```
array:  [0.78539816 1.22474487 1.56123681 1.87082869 2.12132034 2.37904772]
Log value:  [-0.69314718 -0.06931472 -0.01625077 0.12157207 0.38481734  0.52124142]
Absolute value:  [ 0.5  1.5  2.5  3.5  4.5 10.5]
Sine value:  [ 0.47942554  0.99749499  0.57482523 -0.35078322 -0.97753012 -0.62587941]
Cell value:  [ 1.  2.  3.  4.  5. 11.]
Floor value:  [ 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.int8)
Maximum	np.max(arr)
Minimum value	np.min(arr)
Variance	np.var(arr, dtype = np.float32)
Standard Deviation	np.std(arr, dtype = np.float32)
Correlation Coefficient	np.corrcoef(arr1, arr2)

Python3

```
import numpy as np
arr = [10, 7, 4, 3, 2]

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

# summation
print("Sum of arr:", np.sum(arr))

# maximum value
print("Maximum element:", np.max(arr))

# minimum value
print("Minimum element:", np.min(arr))

# variance
print("Variance of arr:", np.var(arr))

# standard deviation
print("Standard deviation of arr:", np.std(arr, dtype = np.float32))

# correlation coefficient
print("Correlation coefficient of arr:", np.corrcoef(arr, arr))
```

Output:

```
Median of arr: 3
Sum of arr: 26
Maximum element: 10
Minimum element: 2
Variance of arr: 10.16
Standard deviation of arr: 3.1875
Correlation coefficient of arr: [[1. 1.]
 [1. 1.]]
```

corrcoef

Python3

```
import numpy as np
arr1 = [1, 2, 3]
arr2 = [4, 5, 6]

# calculate product moment correlation coefficient of the arrays
rcc = np.corrcoef(arr1, arr2)

print(rcc)
```

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 resources.
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, imperfections, and statistical techniques that are crucial. Apart from that, this Numpy Cheat Sheet is thoughtfully organized and categorized, making it easy for beginners to grasp. The NumPy library's ability to handle large datasets efficiently, its support for vectorized operations, and its ease of creating and manipulating arrays make it a cornerstone of data analysis. Mastering data analysis 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 course 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 functions 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.

Don't miss your chance to ride the wave of the data revolution! Every industry is scaling new heights by tapping into the power of data. Sharpen your skills and become a part of the hottest trend in the 21st century.

Dive into the future of technology - explore the Complete Machine Learning and Data Science Program by GeeksforGeeks and stay ahead of the curve.

Last Updated : 18 Mar, 2024

Previous
Git Cheat Sheet

Next
Linux Commands Cheat Sheet

Share your thoughts in the comments

Add Your Comment

Similar Reads

jQuery Cheat Sheet - A Basic Guide to jQuery
Comprehiy Cheat Sheet for Python Operations
Tkinter Cheat Sheet
CSS Cheat Sheet - A Basic Guide to CSS
Linux Commands Cheat Sheet
Git Cheat Sheet
gpstool Cheat Sheet
Subnet Mask Cheat Sheet
C++ STL Cheat Sheet
Python OpenCV Cheat Sheet

aarya_m...

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

Practice Tags : Python

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305
GeeksforGeeks App Store

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

Explore
Job & Internships
All Courses
Mock Tests Series
Hackathon
Geeks Community

Languages
Python
Java
C++
PHP
GoLang
SQL
Machine Learning
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 & AI/ML
Data Science With Python
Data Science For Beginner
Machine Learning Tutorial
ML Maths
Data Visualization Tutorial
Pandas Tutorial
NumPy Tutorial
NLP Tutorial
Deep Learning Tutorial

HTML & CSS
HTML
CSS
Web Templates
CSS Frameworks
Bootstrap
Tailwind CSS
SASS
LESS
Web Design
Design Tutorial

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

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

DevOps
Git
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

Competitive Programming
Top 50 in Algo for CP
Top 50 Tree
Top 50 Graph
Top 50 Array
Top 50 String
Top 50 DP
Top 6 Websites for CP

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

Javascript
JavaScript Examples
Type Script
ReactJS
NextJS
AngularJS
NodeJS
Lodash
Web Browser

Preparation Corner
Company Wise Recruitment Process
Resume Templates
Aptitude Preparation
Puzzles
Company Wise Preparation

School Subjects
Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
World GK

Management & Finance
Management
HR Management
Finance
Income Tax
Organizational Behaviour
Marketing

Free Online Tools
Typing Test
Image Editor
Code Formatters
Code Converters
Currency Converter
Random Number Generator
Random Password Generator

More Tutorials
Software Development
Software Testing
Product Management
SAP
GCC Git Tutorial
Linux
Excel

GeeksforGeeks Videos
DSA
Python
Java
C++
Data Science
CS Subjects

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved