NUMPY ARRAY COMPLETE GUIDE

> *Numpy array is much faster than list and takes less memory but cannot store values of different datatypes*

Import NumPy Library to Create Array

While in list we can directly create and perform operations

```
import numpy as np # np is standard alias for convenience for using numpy operations
```

Delete Function

```
arr_1 = np.array([["s", "p"], ["d", "f"], ['f', 'p']])
del_1 = np.delete(arr_1, 2)
print(del_1)
```

```
['s' 'p' 'f' 'f' 'p']
```

Concept of Matrix

Matrix Creation and Dot Product

```
var1 = np.matrix([[1, 2], [1, 3]])
var2 = np.matrix([[1, 2], [1, 9]])
print(var1)
print(var1.dot(var2))
```

```
[[1 2]
[1 3]]

[[ 3 20]
[ 4 29]]
```

Matrix Functions in NumPy Arrays

1. *Transpose - convert rows into columns*
2. *swapaxes*
3. *inverse*
4. *power*
5. *determinant of matrix*

Transpose Function

```
var1 = np.matrix([[1, 2], [1, 3]])
print(var1)

print(np.transpose(var1)) # transpose
print(var1.T) # transpose shortcut
print(np.swapaxes(var1, 1, 0)) # axis 1 to 0 swapaxes
```

Inverse Function

```
print(np.linalg.inv(var1)) # inverse
```

Matrix Power Function

```
print(np.linalg.matrix_power(var1, 3)) # power when n>0
print(np.linalg.matrix_power(var1, -3)) # inverse*power
when n<0
print(np.linalg.matrix_power(var1, 0)) # gives identity
matrix
```

Determinant Function

```
print(np.linalg.det(var1)) # determinant
```

Handling Missing Values

NaN (Not a Number) - np.isnan() to Detect Null Values

```
arr = np.array([1, 2, 3, np.nan, 9])
print(np.isnan(arr)) # will return True for null value
```

```
[False False False True False]
```

Note: We cannot compare NaN directly: np.nan == np.nan returns False!

Replace NaN Values

```
cleaned_arr = np.nan_to_num(arr, nan=12) # default
value will be zero
print(cleaned_arr)
```

```
[ 1. 2. 3. 12. 9.]
```

Dealing with Infinite Values

Detect Infinite Values

```
arr_1 = ([1, 2, np.inf, 32, -np.inf])
print(np.isinf(arr_1)) # return True or False
```

```
[False False True False True]
```

Replace Infinite Values

```
cleaned_arr1 = np.nan_to_num(arr_1, posinf=199,
neginf=-199)
print(cleaned_arr1)
```

```
[ 1. 2. 199. 32. -199.]
```

Working with CSV Files Using Pandas

Import Pandas and Read CSV

```
import pandas as pd

data = pd.read_csv("Employee.csv")
print(data)
```

Data Analysis Functions

```
# Statistical summary
data.describe()

# Data info
data.info()

# Check missing values
print("Missing values in each column:")
print(data.isnull().sum())
```

Data Cleaning Operations

Fill Missing Values

```
# Fill with mean
data['ExperienceInCurrentDomain'].fillna(data['Experience
inplace=True)

# Fill with mode
data['City'].fillna(data['City'].mode()[0],
inplace=True)

# Fill Age with mean
data['Age'].fillna(data['Age'].mean(), inplace=True)

# Fill EverBenched with mode
data['EverBenched'].fillna(data['EverBenched'].mode()
[0], inplace=True)
```

Drop Missing Values

```
# Drop rows with missing Gender values
data.dropna(subset=["Gender"], axis=0, inplace=True)
```

Handle Infinite Values

```
# Replace infinite values with NaN
data.replace((np.inf, -np.inf), np.nan, inplace=True)
```

Drop Duplicates

```
# Drop duplicate rows based on all columns
data.drop_duplicates(inplace=True)
```

Deal with Negative Values

```
# Replace negative ages with mean
data["Age"] = np.where(data["Age"] < 0,
data["Age"].mean(), data["Age"])
```

Save Cleaned Dataset

```
data.to_csv("cleaned.csv", index=False)
print("Cleaned data and saved successfully!")
```

End of NumPy Complete Guide

Master NumPy for efficient data manipulation and analysis!

Creating Arrays in NumPy

1. Basic 1-Dimensional Array

```
X = np.array([1, 2, 3, 4, 5, 6])
print(X)
```

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

2. Create List and Print as np.array

```
l = []
for i in range(1, 5):
int_i = int(input("Enter an integer"))
l.append(int_i)
print(np.array(l))
```

Multi-Dimensional Arrays

2-Dimensional Array

```
y = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(y)
print(y.ndim)
```

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

3-Dimensional Array

```
Z = np.array([[[1, 2, 3], [4, 5, 6], [7, 8, 9]], [[9, 8, 7], [6,
5, 4], [3, 2, 1]]])
print(Z)
print(Z.ndim)
```

Array Initialization Functions

Arrays Filled with Zeros

```
# 1-dimensional
arr_zero = np.zeros(8)
print(arr_zero)

# 2-dimensional
arr2_zero = np.zeros((2, 4))
print(arr2_zero)

# 3-dimensional
arr3_zero = np.zeros((2, 4, 6))
print(arr3_zero)
```

Arrays Filled with Ones

```
# 1-dimensional
arr_one = np.ones(8)
print(arr_one)

# 2-dimensional
arr2_one = np.ones((2, 4))
print(arr2_one)

# 3-dimensional
arr3_one = np.ones((2, 4, 6))
print(arr3_one)
```

Empty Array

Empty array takes the random updated values in the memory

```
arr_emp = np.empty(4)
print(arr_emp)

arr_emp2 = np.empty((2, 4)) # 2d empty array
print(arr_emp2)

arr_emp3 = np.empty((2, 3, 4)) # 3d empty
print(arr_emp3)
```

Special Array Functions

Full Array - full((shape range), value stored)

```
arr_full = np.full((2, 3), 4) # 2d full
print(arr_full)
```

```
[[4 4 4]
[4 4 4]]
```

Eye/Identity Matrix

```
arr_diag = np.eye(3, 4)
print(arr_diag)
```

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

Linearly Spaced Array

> *Array with values that are linearly spaced in a specified interval*

```
arr_lsp = np.linspace(2, 3, 4)
print(arr_lsp)
```

```
[2. 2.33333333 2.66666667 3. ]
```

Random Functions to Generate Random Values in Array

rand() - Generate values between 0 to 1

```
# 1D array
arr_rd = np.random.rand(5)
```

```
print(arr_rd)

# 2D array
arr_rd_2d = np.random.rand(5, 4)
print(arr_rd_2d)

# 3D array
arr_rd_3d = np.random.rand(5, 2, 2)
print(arr_rd_3d)
```

randn() - Generate numbers closer to 0 (could be negative)

```
arr_rd = np.random.randn(5)
print(arr_rd)

arr_rd_2d = np.random.randn(5, 4)
print(arr_rd_2d)

arr_rd_3d = np.random.randn(5, 2, 2)
print(arr_rd_3d)
```

ranf() - Random numbers in float [0.0 to 1.0)

> *1.0 not included, only takes 1 argument*

```
arr_rd = np.random.ranf(5)
print(arr_rd)
```

randint(min, max, total_values) - Generate random integers

```
arr_rd = np.random.randint(0, 2, 2)
print(arr_rd)
```

Arithmetic Operations in NumPy Arrays

Addition - add(a, b)

```
arr_add = np.array([1, 2, 3, 4, 5])
arr_add2 = np.array([4, 3, 2, 1, 5])
print(arr_add + arr_add2)
```

```
[ 5 5 5 5 10]
```

Subtraction - subtract(a, b)

```
arr_sub = np.array([1, 2, 3, 4, 5])
arr_sub2 = np.array([4, 3, 2, 1, 5])
```

```
print(arr_sub - arr_sub2)
print(np.subtract(arr_sub, arr_sub2))
```

```
[-3 -1 1 3 0]
```

Division - divide(a, b)

```
arr_div = np.array([1, 2, 3, 4, 5])
arr_div2 = np.array([4, 3, 2, 1, 5])
print(arr_div / arr_div2)
print(np.divide(arr_div, arr_div2))
```

Modulus - modulus(a, b)

```
arr_mod = np.array([1, 2, 3, 4, 5])
arr_mod2 = np.array([4, 3, 2, 1, 5])
print(arr_mod % arr_mod2)
print(np.mod(arr_mod, arr_mod2))
```

Reciprocal - reciprocal(1/a)

```
arr_rec = np.array([1, 2, 3, 4, 5])
arr_rec2 = np.array([4, 3, 2, 1, 5])
print(1 / arr_rec)
print(np.reciprocal(arr_rec2))
```

Same operations work with 2D and 3D arrays!

Arithmetic Functions

Min/Max Functions

axis=1 *evaluates the min/max of each column across each row*
axis=0 *(default) evaluates across columns*

```
arr_mod = np.array([[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]])


print(np.min(arr_mod, axis=0), np.argmin(arr_mod)) # argmin/argmax
to find index
print(np.min(arr_mod, axis=1), np.argmin(arr_mod))
print(np.max(arr_mod, axis=1), np.argmax(arr_mod))
print(np.max(arr_mod, axis=0), np.argmax(arr_mod))
```

Square Root Function - sqrt()

```
arr_mod = np.array([[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]])
print(np.sqrt(arr_mod))
```

Trigonometric Functions - cos(), sin()

```
arr_mod = np.array([[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]])
print(np.cos(arr_mod))
print(np.sin(arr_mod))

# Cumulative sum
print(np.cumsum(arr_mod))
```

Shape and Reshape Functions

Shape Function

```
arr_sh = np.array([[1, 2], [1, 2]])
print(arr_sh)
print(f'shape of array {arr_sh.shape}')
```

```
[[1 2]
[1 2]]
shape of array (2, 2)
```

Creating Multi-Dimensional Arrays with ndim

```
arr_sh = np.array([1, 2, 1, 2], ndim=4)
print(arr_sh)
print(f'shape of array {arr_sh.shape}')
```

```
[[[[1 2 1 2]]]]
shape of array (1, 1, 1, 4)
```

Reshape Function

```
arr_sh1 = np.array([1, 2, 1, 2, 4, 6, 8, 3, 5])
print(arr_sh1.ndim) # Output: 1
print(arr_sh1)

X = arr_sh1.reshape(3, 3)
print(X)
print(X.ndim) # Output: 2
```

Reshape to 3D and Back to 1D

```
arr_sh3 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

X1 = arr_sh3.reshape(2, 3, 2)
print(X1)
print(X1.ndim) # Output: 3

# Convert n to 1 dimension arrays
one_dim = arr_sh3.reshape(-1)
print(one_dim)
print(one_dim.ndim) # Output: 1
```

Broadcasting

> *Broadcasting will be performed on 2x1 with 2x1 or 2x2... Arrays must have compatible dimensions*

Broadcasting Example 1

```
var = np.array([1, 2, 3])
var2 = ([1], [2], [3])
print(var * var2)
```

```
[[1 2 3]
[2 4 6]
[3 6 9]]
```

Broadcasting Example 2

```
var1 = np.array([[1, 2, 3], [1, 2, 3], [1, 2, 3]])
var3 = np.array([1, 2, 3])
print(var1 * var3)
```

```
[[1 4 9]
[1 4 9]
[1 4 9]]
```

Slicing Arrays

Basic Slicing

```
var = np.array([7, 6, 5, 4, 3, 3])
print(var[::2]) # print values skipping 1 value in between
print(var[::-1]) # reverse an array
```

```
[7 5 3]
[3 3 4 5 6 7]
```

2D Array Slicing

```
var2 = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(var2[1, :]) # print all the elements of row 2
```

```
[5 6 7 8]
```

3D Array Slicing

```
var3 = np.array([[[1, 2, 3, 4, 5], [24, 56, 6, 7, 8]], [[9, 8, 7,
6, 5], [3, 4, 6, 7, 1]]])
print(var3[1, 0, 3])
```

```
6
```

Iteration

1D Array Iteration

```
var = np.array([7, 6, 5, 4, 3, 3])
for i in var:
print(i)
```

2D Array Iteration with Nested Loops

```
var2 = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
for i in var2:
for l in i:
print(l)
```

3D Array Iteration

```
var2 = np.array([[[1, 2, 3, 4], [5, 6, 7, 8]], [[2, 3, 1, 4], [3,
7, 6, 4]]])
for i in var2:
for l in i:
```

```
for k in l:
print(k)
```

Using nditer() - Avoid Multiple For Loops

```
var2 = np.array([[[1, 2, 3, 4], [5, 6, 7, 8]], [[2, 3, 1, 4], [3, 7, 6, 4]]])
for i in np.nditer(var2):
print(i)
```

Row Major (C) and Column Major (F) Order

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

for element in np.nditer(arr, order='F'): # Column major order
print(element)
```

```
1
3
2
4
```

ndenumerate() - Get Index and Value

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

for element in np.ndenumerate(arr):
print(element)
```

```
((0, 0), 1)
((0, 1), 2)
((1, 0), 3)
((1, 1), 4)
```

Formatted ndenumerate Output

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

for index, value in np.ndenumerate(arr):
print(f"Index: {index}, Value: {value}")
```

Type Conversion During Iteration

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

for x in np.nditer(arr, flags=['buffered'], op_dtypes=['S']):
print(x)
```

```
b'1'
b'2'
b'3'
```

nditer - Modifying Array Values

Enables in-place modification of array elements by specifying readwrite or writeonly flags for operands. When modifying, the nditer should be used as a context manager (with statement) or its close() method should be called to ensure changes are written back to the original array.

```
arr = np.array([[1, 2], [3, 4]])
print("Original arr:")
for i in arr:
print(i)

print("Modification in original array")
with np.nditer(arr, op_flags=["readwrite"]) as it:
for k in arr:
k[...] = k * 2 # modify in place original array
print(arr)
```

Copy vs View

> *Copy means creating a new array and view means viewing the original array. Any changes made in view will be made in original array.*

Copy Example

```
arr = np.array([[1, 2], [3, 4]])
cop = arr.copy()
print("original:", arr)
print("copy:", cop)
```

View Example - Changes Affect Original

```
arr = np.array([[1, 2], [3, 4]])
vw = arr.view()
with np.nditer(vw, op_flags=['readwrite']) as it:
for i in it:
i[...] = i * 2
print("original:", arr) # Changes reflected!
print("view:", vw)
```

Joining Arrays

> *Using 1. concatenate, 2. stack (hstack, vstack, dstack, axis=0, axis=1)*

Concatenate Along Different Axes

```
arr = np.array([[1, 2], [3, 4]])
arr_1 = np.array([[0, 2], [8, 5]])
```

```
print(np.concatenate((arr, arr_1), axis=1)) # along the row
print(np.concatenate((arr, arr_1), axis=0)) # along the column
```

Stack Functions

```
arr = np.array([[1, 2], [3, 4]])
arr_1 = np.array([[0, 2], [8, 5]])

print(np.stack((arr, arr_1), axis=1))
print(np.vstack((arr, arr_1))) # vertical stack (column)
print(np.hstack((arr, arr_1))) # horizontal stack (row)
print(np.dstack((arr, arr_1))) # depth stack (height)
```

Split Array

Split 1D Array

```
arr = np.array([1, 2, 3, 4])
print(arr)
print(np.split(arr, 2))
```

```
[1 2 3 4]
[array([1, 2]), array([3, 4])]
```

Split 2D Array

```
arr = np.array([[1, 2], [3, 4], [3, 5]])
print(arr)
print(np.split(arr, 2, axis=1))
```

NumPy Array Functions

Search Function - where()

```
# 1D Array Search
arr = np.array([1, 2, 3, 4, 3, 5])
print(np.where(arr == 3)) # search and return indexes
```

```
(array([2, 4], dtype=int64),)
```

```
# 2D Array Search
arr = np.array([[1, 2], [3, 4], [3, 5]])
print(np.where(arr == 3)) # returns row and column indices
```

searchsorted() - Binary Search Style

> *Works like a binary array, gives the index where any new particular value could be placed*

```
arr = np.array([1, 2, 3, 5])
print(np.searchsorted(arr, 4))
print(np.searchsorted(arr, 4, side="right"))
```

Sort Function

```
# Sort numbers
arr = np.array([1, 23, 4, 5, 6, 3, 52, 0])
print(np.sort(arr))
```

```
[ 0 1 3 4 5 6 23 52]
```

```
# Sort strings
arr_1 = np.array(["s", "p", "d", "f"])
print(np.sort(arr_1))
```

```
['d' 'f' 'p' 's']
```

```
# Sort 3D array element
arr = np.array([[[1, 23], [41, 5]], [[6, 3], [52, 0]]])
print(np.sort(arr[0, 1]))
```

Filter Array

```
arr_1 = np.array(["s", "p", "d", "f"])
f = [True, False, True, False]
print(f)
new_n = arr_1[f]
print(new_n)
```

```
[True, False, True, False]
['s' 'd']
```

Additional Array Functions

Shuffle Function

```
arr_1 = np.array(["s", "p", "d", "f"])
np.random.shuffle(arr_1)
print(arr_1)
```

Unique Function

```
arr_1 = np.array(["s", "p", "d", "f", 'f', 'p'])
np.unique(arr_1)
print(np.unique(arr_1, return_index=True, return_counts=True))
```

Resize Function

```
arr_1 = np.array(["s", "p", "d", "f", 'f', 'p'])
np.resize(arr_1, (2, 3))
```

Flatten / Ravel

> *Convert two-dimensional array into 1 dimension. Order could be C, F, A, or K.*

```
arr_1 = np.array(["s", "p", "d", "f", 'f', 'p'])
y = np.resize(arr_1, (2, 3))
print(y)

print("Flatten (method of numpy):", y.flatten(order="F"))
print("Ravel (parameter of Numpy):", np.ravel(y, order="F"))
```

Insert and Append

Insert Function

```
arr_1 = np.array(["s", "p", "d", "f", 'f', 'p'])
arr = np.insert(arr_1, 3, "e")
print(arr_1)
print(arr)
```

Insert in 2D Array - Axis 0 (Vertically)

> Axis 0: This axis runs vertically, down the rows. Operations performed along axis=0 will apply independently to each column.

```
arr_1 = np.array([["s", "p"], ["d", "f"], ['f', 'p']])
arr = np.insert(arr_1, 3, "e", axis=0)
ar = np.append(arr_1, [["e", "y"]], axis=0)
print(arr_1)
print(arr)
print(ar)
```

Insert in 2D Array - Axis 1 (Horizontally)

> Axis 1: This axis runs horizontally, across the columns. Operations performed along axis=1 will apply independently to each row.

```
arr_1 = np.array([["s", "p"], ["d", "f"], ['f', 'p']])
arr = np.insert(arr_1, (1, 2), ["e", "y"], axis=1)
ar = np.append(arr_1, [["e", "y"]])
print(arr_1)
print(arr)
print(ar)
```