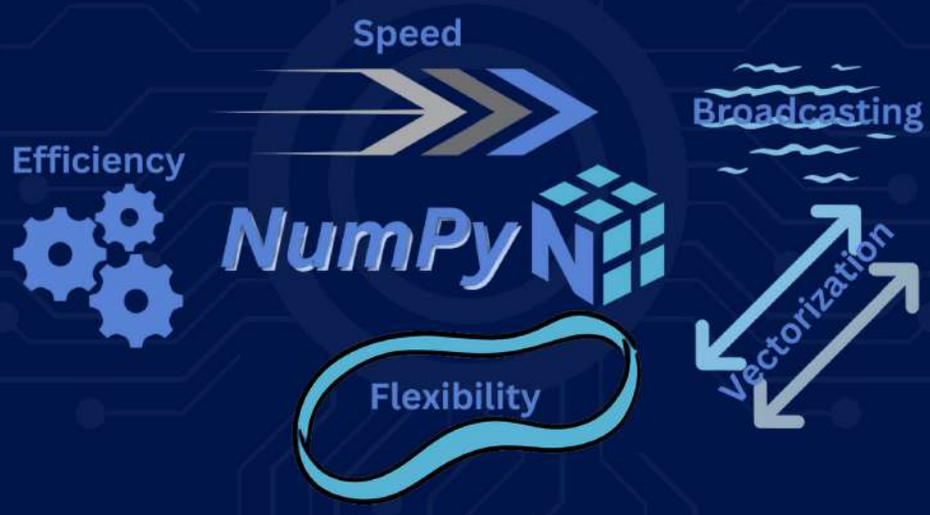
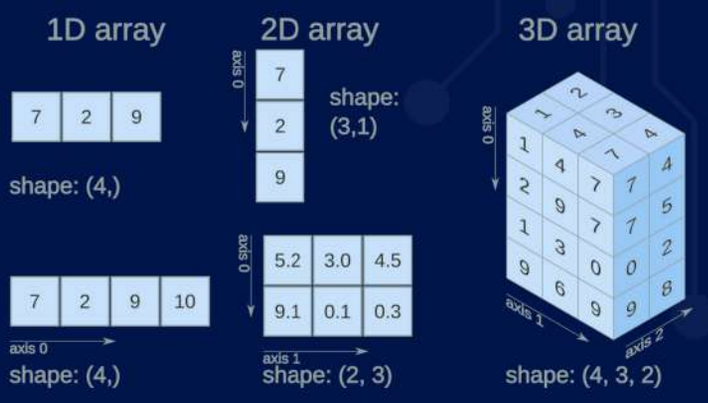
NumPy

https://bit.ly/np_practice

NUMPY PRACTICE

Exploring the Depths of NumPy: A Comprehensive Journey Through Python's Essential Library





1 Numpy

- NumPy is a Python library
- NumPy is used for working with arrays
- NumPy is Short for “Numerical Python”

[ ]:
```
# Import numpy library
import numpy as np

# Example of Numpy
x = np.array([1,2,3,4])
print(x)
print(type(x))
```

```
[ 1 2 3 4
<class 'numpy.ndarray'>
```

[ ]:
```
# Example of List
y = [1,2,3,4]
print(y)
print(type(y))
```

```
[1, 2, 3, 4]
<class 'list'>
```

2 Numpy vs List

- **Data Types Storage**

list = [1, True, ”Hello] array

(only one data type) arr =

[1,2,3,4]

- importing module
- numerical operation
- modification capabilities
- consume less memory
- Fast as compared to the python list
- Convenient to use

3 Array dimension

[ ]:
```
# Take user input and create an array
l = []

for i in range(1,5):
  int_1 = int(input("Enter : "))
  l.append(int_1)
print(np.array(l))
```

```
Enter : 1
Enter : 2
Enter : 4
Enter : 5
[ 1 2 4 5
```

[ ]:
```
# check the dimension of array

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

```
1
```

[ ]:
```
# 2 Diamension array

arr2 = np.array([[1,2,3,4],[1,4,5,6]])
print(arr2)
print("Diamension of numpy array", arr2.ndim)
```

```
[[1 2 3 4]
 [1 4 5 6]]
Diamension of numpy array 2
```

[ ]:
```
# 3 Diamension array

arr3 = np.array([[[1,2,3,4],[4,5,8,7],[4,5,2,1]]])
print(arr3)
print("Number of Diamesion: ",arr3.ndim)
```

```
[[[1 2 3 4]
  [4 5 8 7]
  [4 5 2 1]]]
Number of Dimensions: 3
```

[ ]:
```
# Multiple Dimension array
arm = np.array([1,2,3], ndmin = 10)
print(arm)
print(arm.ndim)
```

```
[[[[[[[[[[1 2 3]]]]]]]]]]
10
```

4 Special Numpy array

- Array filled with 0's
- Array filled with 1's
- Create an empty array
- An array with a range of elements
- Array diagonal element filled with 1's
- Create an array with values that are spaced linearly in a specified interval

4.0.1 Zero Array

[ ]:
```
import numpy as np

ar_zero = np.zeros(4)
ar_zero1 = np.zeros((3,4))

print(ar_zero1)
print()

print(ar_zero)
```

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

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

4.0.2 Ones Array

[ ]:
```
ar_ones = np.ones(4)

print(ar_ones)
```

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

4.0.3 Empty Array

[ ]:
```
ar_em = np.empty(4)

print(ar_em)
```

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

4.0.4 Range

[ ]:
```
arr_rn = np.arange(4)
print(arr_rn)
```

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

4.0.5 Diagonal

[ ]:
```
ar_dia = np.eye(3)
print(ar_dia)
```

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

[ ]:
```
# create 3*5 diagonal matrix

dia_m = np.eye(3,5)
print(dia_m)
```

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

4.0.6 Linespace

[ ]:
```
# Create an array with values that are spaced linearly in a
specified interval ar_lin = np.linspace(0,20,num=5) print(ar_lin)
```

```
[ 0. 5. 10. 15. 20.]
```

5 Random

[ ]:
```
# rand() --> Generate random value b/w 0-1

import numpy as np
var = np.random.rand(4)
print(var)
```

```
[0.75555822 0.94566929 0.01913765 0.94010246]
```

[ ]:
```
var1 = np.random.rand(2,5)
print(var1)
```

```
[[0.39158461 0.02826316 0.46298353 0.48599032 0.95100693]
 [0.7050555 0.72144787 0.36630705 0.23798353 0.35968228]]
```

```
[ ]: # randn() --> The function is used to generate a random value close
to zero
     # negative or positive
     value var2 =
     np.random.randn(5)
     print(var2)
```

```
[-2.06166922 -0.80890127 -0.21155592 0.89337184 -0.52748609]
```

```
[ ]: # ranf() --> random float in the half-open interval [0.0, 1.0]

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

```
[0.19137554 0.29957276 0.0558102 0.58667686 0.65355229]
```

```
[ ]: # randint(min, max, total_value) --> Generate number b/w a range

     var = np.random.randint(5,50,5)
     print(var)
```

```
[42 16 43 20 38]
```

6 Data Types in Numpy

```
[ ]: # Int Data type
     import numpy as np

     var = np.array([1,2,3,4])
     print("Data Type : ", var.dtype)
```

```
Data Type :  int64
```

```
[ ]: # float Data type

     var = np.array([1.2,1.0,1.5,1.6])
     print("Data Type : ", var.dtype)
```

```
Data Type :  float64
```

```
[ ]: # String Data Type

     var = np.array(["a","b"])
     print("Data Type : ", var.dtype)
```

```
Data Type :  <U1
```

[ ]:
```
# change the data type

var = np.array([1,2,3,4], dtype = "f")
print(var)
print(var.dtype)
```

```
[1. 2. 3. 4.]
float32
```

[ ]:
```
# Convert the data type

x2 = np.array([1,2,3,4])

new = np.float32(x2)

print(x2.dtype)
print(new.dtype)
```

```
int64
float32
```

7 Arithmetic Operation

[ ]:
```
import numpy as np

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

# Add 3 with all elements
varadd = var+3
print(varadd)
```

```
[ 4 5 6 7
```

[ ]:
```
# Add two numpy array
var1 = np.array([1,2,3,4])
var2 = np.array([1,2,3,4])

addArrays = var1+var2
print(addArrays)
```

```
[ 2 4 6 8
```

[ ]:
```
# Subtraction of array

var1 = np.array([2,4,6,8])
var2 = np.array([1,2,3,4])

subArrays = var1-var2

print(subArrays)
```

```
[ 1 2 3 4
```

[ ]:
```
# Multiply
var1 = np.array([2,4,6,8])
var2 = np.array([1,2,3,4])

subArrays = var1*var2
print(subArrays)
```

```
[ 2  8 18 32]
```

[ ]:
```
# Divide
var1 = np.array([2,4,6,8])
var2 = np.array([1,2,3,4])

subArrays = var1/var2
print(subArrays)
```

```
[2. 2. 2. 2.]
```

[ ]:
```
# Modulus
var1 = np.array([2,9,7,10])
var2 = np.array([1,2,3,4])

subArrays = var1%var2
print(subArrays)
```

```
[ 0 1 1 2
```

[ ]:
```
# Using numpy add the arrays

var1 = np.array([2,4,6,8])
var2 = np.array([1,2,3,4])

AddNp = np.add(var1, var2)
AddNp
```

[ ]: array([3, 6, 9, 12])

7.0.1 Numpy Arithmatic Operations

- a+b –> np.add(a,b)
- a-b –> np.subtract(a,b)
- a*b –> np.multiply(a,b)
- a/b –> np.divide(a,b)
- a%b –> np.mod(a,b)
- a**b –> np.power(a,b)
- 1/a –> reciprocal(a)

[ ]:
```
# Arithmetic operation with 2d Arrays

var1 = np.array([[1,2,3,4],[1,2,3,4]])
var2 = np.array([[2,3,4,5],[4,5,6,7]])

print(var1)
print()
print(var2)
print()

Add2D = np.add(var1, var2)
print(Add2D)
```

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

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

[[ 3 5 7 9]
 [ 5 7 9 11]]
```

8 Arithmetic Numpy functions

- np.min(x)
- np.max(x)
- np.argmin(x)
- np.sqrt(x)
- np.sin(x)
- np.cos(x)
- np.cumsum(x)

[ ]:
```
import numpy as np

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

```
print("Min number and Position: ", min(var),
np.argmin(var)) print("max number and position: ",
max(var), np.argmax(var)) Min number and Position: 1 0
max number and position:  5 4
```

[ ]:
```
# 2D Array
# axis = 0 --> work according to col
# axis = 1 --> row
var = np.array([[5,2,3,4],[5,6,4,1]])
print(np.min(var, axis=1))
```

```
[2 1]
```

[ ]:
```
# axis = 0 --> col
var = np.array([[5,2,3,4],[5,6,4,1]])
print(np.min(var, axis=0))
```

```
[ 5 2 3 1
```

[ ]:
```
# Square root

var = np.array([[5,2,3,4],[5,6,4,1]])
print("sqrt : ", np.sqrt(var))
```

```
sqrt : [[2.23606798 1.41421356 1.73205081 2.     ]
  [2.23606798 2.44948974 2.       1.       ]]
```

[ ]:
```
# sign and cos value

var2 = np.array([1,2,3])
print(np.sin(var2))
print(np.cos(var2))
```

```
[0.84147098 0.90929743 0.14112001]
[ 0.54030231 -0.41614684 -0.9899925 ]
```

[ ]:
```
# cumsum() --> Add continue with previous value

var = np.array([2,4,5,6])
print(np.cumsum(var))
```

```
[ 2  6 11 17]
```

[ ]:

9 Shape and Reshape

```
# check diamention of array
import numpy as np

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

print(var)
print()
print(var.shape)
```

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

[ ]:
```
# Multi diamensional Array
var1 = np.array([1,2,3,4], ndmin=4)

print(var1)
print(var1.ndim)
print()
print(var1.shape)
```

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

[ ]:
```
# reshape --> Change the 1d array to multi diamension array

var2 = np.array([1,2,3,4,5,6])

print("1D Array: ", var2)
print("Number of diamension: ", var2.ndim)
print()

# reshape()
x = var2.reshape(2,3)
print("After Reshape \n", x)
print("Number of diamension: ",  .ndim)
```

```
1D Array:  [ 1 2 3 4 5 6 ]
Number of diamension: 1

After Reshape
 [[1 2 3]
 [4 5 6]]
Number of diamension: 2
```

[ ]:
```
# reshape reverse
x1 = x.reshape(-1)
print(x1)
```

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

10 Broadcasting

[ ]:
```
# During addition time if array size is tont same then this wrroe occurs

import numpy as np

var1 = np.array([1,2,3,4])
var2 = np.array([1,2,3])

print(var1.shape)

print(var1+var2)
```

```
(4,)
```

```
---------------------------------------------------------------------------
-------
ValueError                                Traceback (most recent call last)
<ipython-input-4-0bc9a560946f> in <cell line: 10>()
      8 print(var1.shape)
      9
---> 10 print(var1+var2)

ValueError: operands could not be broadcast together with shapes
(4,) (3,)
```

[ ]:
```
var1 = np.array([1,2,3,4])
var2 = np.array([[1],[2],[3],[4]])

print("Shape of var1 ", var1.shape)
print("Shape of var2 ", var2.shape)

sum = var1+var2
print(sum)
```

```
Shape of var1  (4,)
Shape of var2 (4, 1)
[[2 3 4 5]
 [3 4 5 6]
 [4 5 6 7]
 [5 6 7 8]]
```

[ ]:

11 Indexing and slicing

```
import numpy as np

var = np.array([1,2,3,4])
#       index - 0,1,2,3
#  rev Index - -4,-3,-2,-1

print(var[-3])
```

```
2
```

[ ]:
```
# indexing in 2D array

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

print(var[1,2])
```

```
2
7
```

[ ]:
```
# Slicing
import numpy as np
var = np.array([1,2,3,4,5,6,7])

print(var)

print("Value 5 to 7 ", var[5:7])
print("Steps : ", var[1:6:2])
```

```
[ 1 2 3 4 5 6 7 ]
Value 5 to 7 [6 7]
Steps : [2 4 6]
```

[ ]:
```
 # Slicing in 2d array
var = np.array([[1,2,3,4],[5,6,7,8],[8,5,6,2]])
print("2nd row last 2 element: ",var[2,-2:])
```

```
2nd row last 2 element: [6 2]
```

12 Iterating in numpy

[ ]:
```
# 1D Array

import numpy as np
```

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

for i in var:
  print(i)
```

```
1
2
3
4
5
```

[ ]:
```
# iteration in 2D array

var1 = np.array([[1,2,3,4],[5,6,7,8]])
for i in var1:
  for j in i:
    print(j)
```

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

[ ]:
```
# Using nditer to iterate the elements

var1 = np.array([[1,2,3,4],[5,6,7,8]])

for i in np.nditer(var1):
  print(i)
```

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

[ ]:

13 Copy vs View

```
import numpy as np

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

# Using copy()
co = var.copy()

print("Var:",var)
print("Co", co)
```

```
Var: [1 2 3 4]
C o [ 1 2 3 4
```

[ ]:
```
# Using vuew()

var1 = np.array([1,2,3,4,5])

vw = var.view()

print("View", vw)
```

```
View [1 2 3 4]
```

14 Join 2 Array

[ ]:
```
var1 = np.array([[1,2],[3,4]])
var2 = np.array([[4,5],[7,8]])

vr = np.concatenate((var1, var2), axis=0)
print(vr)
```

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

15 Search

[ ]:
```
import numpy as np

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

# index search
x = np.where(var==2)
```

```
print(x)

# using %
x = np.where((var%2) == 0)

print(x)
```

```
(array([1, 5, 9]),)
(array([ 1,  3,  5,  9, 10]), )
```

[ ]:
```
# Search Sorted Array

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

x1 = np.searchsorted((var%2), 5)
print(x1)
```

```
5
```

[ ]:
```
# sort array

var = np.array([1,2,5,8,54,23,52,12,45,14])
n = np.sort(var)
n
```

[ ]:
```
array([ 1,  2,  5,  8, 12, 14, 23, 45, 52, 54])
```

[ ]:
```
# Charater sorting using numpy

c = np.array(['s','c','d','t'])

sorted_char = np.sort(c)

sorted_char
```

[ ]:
```
array(['c', 'd', 's', 't'], dtype='<U1')
```

[ ]:
```
# Filter Array
var3 = np.array([1,5,7,8])
f = [True, False, True, True]

new_a = var3[f]
print(new_a)
```

```
[1 7 8]
```

[ ]:

16 Numpy Array Function –> Arithmatic Functions

```
# Shuffle Function
import numpy as np

var = np.array([1,2,3,4,5])
np.random.shuffle(var)

print(var)
```

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

[ ]:
```
# Unique Function
var1 = np.array([1,2,6,3,4,5,4,2,3,1])

n = np.unique(var1, return_index = True)

print(n)
```

```
(array([1, 2, 3, 4, 5, 6]), array([0, 1, 3, 4, 5,
2]))
```

[ ]:
```
# resize function
var1 = np.array([1,2,6,3,4,5,4,2,3,1])

y = np.resize(var1,(2,3))
print(y)

# Convert in flatten array
print(y.flatten())
```

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

17 Insert and Delete function in Array

[ ]:
```
import numpy as np

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

inserted_value = np.insert(var, 2,  12,45,74])
print("After Inserting: ", inserted_value)
```

```
[1 2 3 5 4 6]
After Inserting: [ 1 2 12 45 74 3 5 4 6]
```

[ ]:
```
# Insert data in 2D Array

var1 = np.array([[1,2,3],[4,5,6]])

v1 = np.insert(var1, 2,  22,23,25], axis=0)

print(v1)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [22 23 25]]
```

18 Matrix vs Array

[ ]:
```
# dot product of matrix
import numpy as np

var = np.matrix([[1,2],[1,4]])
var2 = np.matrix([[1,2],[1,2]])
print(type(var))

print()
print(var.dot(var2))

print("Without bot product: ", var*var2)
```

```
<class 'numpy.matrix'>

[[ 3  6]
 [ 5 10]]
Without bot product: [[ 3  6]
 [ 5 10]] [ ]:
```