Numpy Programming Lab

```
# 1
!pip install numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.23
```

```
# 2

import numpy as np
```

```
# 3

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

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

```
# 4

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

```
(2, 3)
```

```
# 5

arr2=np.array([[[1,2,3]]])
print(arr2.shape)
```

```
(1, 1, 3)
```

```
# 6

arr= np.array([1,2,3,4], ndmin=3) # 3D array created using ndmin
print(arr)
print(arr.shape)

arr= np.array([1,2,3,4], ndmin=5) # 5D array created using ndmin
print(arr)
print(arr.shape)
```

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

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

```
# 7 Numpy array with random values

Data = np.random.random((3,4))
print("The array is:\n", Data)
print(type(Data))
print(np.ndim(Data))
print(Data.size)
print(Data.shape)
```

```
The array is:
 [[0.93613865 0.42445364 0.52029089 0.46929546]
 [0.33137612 0.9693958  0.85476669 0.81680548]
 [0.5241322  0.14721576 0.4027964  0.18530804]]
<class 'numpy.ndarray'>
2
12
(3, 4)
```

```
# 8 Numpy array of zeros

Data = np.zeros((3, 4))
print("The array is:\n", Data)
print(type(Data))
print(np.ndim(Data))
print(Data.size)
print(Data.shape)
```

```
The array is:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
<class 'numpy.ndarray'>
2
12
(3, 4)
```

```
# 9 Numpy array of ones

Data = np.ones((2, 3))
print("The array is:\n", Data)
print(type(Data))
print(np.ndim(Data))
print(Data.size)
print(Data.shape)
```

```
The array is:
 [[1. 1. 1.]
 [1. 1. 1.]]
<class 'numpy.ndarray'>
2
```

```
6
(2, 3)
```

```
# 10 Empty numpy array

Data = np.empty((3, 4))
print("The array is:\n", Data)
print(type(Data))
print(np.ndim(Data))
print(Data.size)
print(Data.shape)
```

```
The array is:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
<class 'numpy.ndarray'>
2
12
(3, 4)
```

```
# 11 Array with range of elements

Data = np.arange(12).reshape(3, 4)
print("The array is:\n", Data)
print(type(Data))
print(np.ndim(Data))
print(Data.size)
print(Data.shape)
```

```
The array is:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
<class 'numpy.ndarray'>
2
12
(3, 4)
```

```
# 12 numpy array using linspace

var_arr10=np.linspace(2,25,num=5) # starting from 2 to 25 it genrates 5 no's who are equa
print(var_arr10)
```

```
[ 2.    7.75 13.5  19.25 25.  ]
```

```
# 13

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

```
(2, 3)
```

```
# 14 Number of elements in an array

print(arr.size)  # arrayname,size is used to return number of elements in an array
```

```
6
```

```
# 15

print(np.ndim(arr)) #get the dimension of arr
```

```
2
```

```
# 16

arr= np.array([1,2,3],dtype=int)
print(arr)

arr_type=np.float64(arr)
print(arr_type)
```

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

```
# 17
arr= np.array([1,2,3],dtype=float)
print(arr)

arr_type=arr.astype(int)
print(arr_type)
```

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

```
# 18

arr=np.array([3,7,1,8,4,6,0,2,5])
min_value= np.min(arr,axis = 0)
print(min_value)
max_value= np.max(arr,axis = 0)
print(max_value)
```

```
0
8
```

```
# 19

arr=np.array([3,7,1,8,4,6,0,2,5])
Data = arr.reshape(3, 3) #Reshaping the arr array to 3D array
print(Data)
```

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

```
# 20

arr=[1,2,3,4,5,6,7,8]
new_arr=np.array_split(arr,3)
print(new_arr)
```

```
[array([1, 2, 3]), array([4, 5, 6]), array([7, 8])]
```

```
# 21

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

arr_h_split=np.array([[1, 2, 3, 4],
                          [5, 6, 7, 8],
                          [9, 10, 11, 12]])
print(np.vsplit(arr_v_split,2))
print(np.hsplit(arr_h_split,2))
```

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

```
# 22  adiing new dimension

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

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

```
# 23  flattening arrays

flat_array=new_arr.flatten()
print(flat_array)
```

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

```
# 24 Sort

arr=np.array([3,7,1,8,4,6,0,2,5])
np.sort(arr)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
# 25 Concatenate

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

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

```
# 26
x=np.array([[1,2],[3,4]])
y=np.array([[5,6],[7,8]])
result_add=x+y
print(result_add)
```

```
[[ 6  8]
 [10 12]]
```

```
# 27

x=np.array([1,2,3,4])
y=np.array([5,6,7,8])
result_power=np.power(y,x)  #y as a power of x
print(result_power)
```

```
[   5   36  343 4096]
```

```
# 28

x=np.array([1,2,3])
mean_value=np.mean(x)
median_value=np.median(x)
sum_value=np.sum(x)
print("Mean:",mean_value)
print("Median:",median_value)
print("Sum:",sum_value)
```

```
Mean: 2.0
Median: 2.0
Sum: 6
```

```
# 29

x=np.array([[1,2],[3,4]])
y=np.array([[5,6],[7,8]])
result_mat_mul=np.dot(x,y)
print("Matrix Mul: ")
print(result_mat_mul)
```

```
Matrix Mul:
[[19 22]
 [43 50]]
```

```
# 30

x=np.array([[1,2],[3,4]])
transpose_mat=np.transpose(x)
print(transpose_mat)
```

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

31

np.dot function is used to perform matrix multiplication on arrays in numpy

```
# 32

x = np.array([[1,2],[3,4]])
y = np.linalg.inv(x) # inversion of x done in y
# print(x)
# print(y)
# print(np.dot(x,y))
```

```
[[1 2]
 [3 4]]
[[-2.   1. ]
 [ 1.5 -0.5]]
[[1.0000000e+00 0.0000000e+00]
 [8.8817842e-16 1.0000000e+00]]
```

```
# 33 linear algebra operations

arr1=np.array([[1,2,3],[4,5,6],[7,8,9]])
arr2=np.array([[11,12,13],[4,5,6],[17,18,19]])
new1=np.matmul(arr1,arr2)
print(new1)
```

```
[[ 70  76  82]
 [166 181 196]
 [262 286 310]]
```

34

Missing values or nan values can be done by using the function "numpy.isnan()" it will give us the indexes which are having nan values and when combined with other function which is "numpy.logical_not()" where the boolean values will be reversed.

```
# 34

a = np.array([[1,np.nan,3,2,5],[6,8,7,np.nan,10]])
print(np.isnan(a))
```

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

```
# 35 explore np.where

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

```
(array([0]), array([2]))
```

```
# 36 Element wise multiplication

arr1=np.array([[1,2,3],[4,5,6],[7,8,9]])
arr2=np.array([[11,12,13],[4,5,6],[17,18,19]])
new=np.multiply(arr1,arr2)
print(new)
```

```
[[ 11  24  39]
 [ 16  25  36]
 [119 144 171]]
```

```
# 37 Calculate mean and median

arr1=np.array([[1,2,3],[4,5,6]])
j=1
for i in arr1:
  print(j,"row meadian is ",np.median(i)," mean is ",np.mean(i))
  j+=1
```

```
1 row meadian is  2.0  mean is  2.0
2 row meadian is  5.0  mean is  5.0
```

```
# 38  Standard deviation

arr1=np.array([3,7,1,8,4,6,0,2,5])
print(np.std(arr1))
```

```
2.581988897471611
```

39

copy(view) has a base while the deep copy(copy) has no base since it is a copy of base array.Changes made in base array cant be viewed in a deep copy(copy) while the base can be viewed in a view(shallow copy).

```
# 39 copy vs views

ar=np.array([1,2,3])
print(ar)
x=ar.copy()     # copy created
print(x)

y=ar.view()     # view created
print(y)
y[1]=45

print(y)
print(ar)
```

```
[1 2 3]
[1 2 3]
[1 2 3]
[ 1 45  3]
[ 1 45  3]
```

```
# 40 Slicing for ecxtracting sub array

arr=np.array([[1,2,3],[4,5,6],[7,8,9]])
sub= arr[0:2,:]
print(sub)
```

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

```
#41 Indexing

y=np.array([10,20,30,40,50])
print(y[4]) # value at index 4 of y
```

```
50
```

```
#42 How to slice elements

er=np.array([1,2,3,4,5])
sl=er[:4]  # slicing from start to 4th element (4th index is not inclusive)
print(sl)
```

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

```
#43 select every nth element

nth=np.array([1,2,3,4,5,6,7,8,9,10])
sl=nth[::2]
print(sl)
```

```
[1 3 5 7 9]
```

```
#44 extract specific column

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
col = arr[:, 1] #extracting second column
print(col)
```

```
[2 5 8]
```

```
#45 Reversing of rows

import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
arr_reversed = np.flip(arr, axis=0)  # reversing using flip function
print(arr_reversed)
```

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

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

```
#46 Extracting diagonal elements

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

```
     [1 5 9]
```

47

In NumPy, boolean arrays are NumPy arrays with array components that are either “True” or “False” 1. Slicing in NumPy is performed in the same way as it is performed in Python lists 23. To slic a NumPy array using a boolean array, we can use the boolean array as a mask to filter the elements of the original array. For example, if we have an array a and a boolean array b, we can slice a using b as follows: a[b] This will return a new array that contains only the elements of a that correspond to True values in b 4.

```
#47 Slicing with Boolean arrays
ar=np.array([True,False,False,True,True])
print(ar)
print(ar[1:])
```

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

```
#48
#To select elements based on a condition in Python,
#we can use list comprehension to filter the list and create a new list with only the des


arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
result = arr[arr > 5]
print(result)
```

```
     [ 6  7  8  9 10]
```

```
#49 Slice a 3D Numpy array

arr_3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
sliced_arr = arr_3d[0:2, 0:2, 0:2]
print(sliced_arr)
```

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

```
       [[ 7  8]
        [10 11]]]
```

```
#50 Extract specific rows based on condition

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

```
     [1 2 3]
```

51

"'The term broadcasting describes how NumPy treats arrays with different shapes during arithmetic operations. Subject to certain constraints, the smaller array is “broadcast” across the larger array so that they have compatible shapes'"

```
#52 Saving numpy array to a file and loading it back

with open('test.npy', 'wb') as f:
    np.save(f, np.array([1, 2]))
    np.save(f, np.array([1, 3]))
with open('test.npy', 'rb') as f:
    a = np.load(f)
    b = np.load(f)
print(a, b)
```

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

```
# 53

# You can check if two NumPy arrays are equal element-wise using the == operator.
#  When you use the == operator on two NumPy arrays, it performs element-wise
#  comparison and returns a boolean array of the same shape, where each element
#   indicates whether the corresponding elements in the two arrays are equal or n

# Import numpy
import numpy as np

arr = np.array([2,4,5,7,9])
print("First array:", arr)
arr1 = np.array([2,4,6,7,10])
print("Second array:", arr1)

# Check array equal element wise
# Using == operator
result = arr == arr1
print("Check the element-wise equality:", result)
```

```
First array: [2 4 5 7 9]
Second array: [ 2  4  6  7 10]
Check the element-wise equality: [ True  True False  True False]
```

```
# 54. USE OF AND, OR, NOT

arr_and = np.arange(5)
print(np.logical_and(arr_and>1, arr_and<4))
print(np.logical_or(arr_and>1, arr_and<4))
print(np.logical_not([True, False, 0, 1]))
```

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

```
#55
# The numpy.meshgrid function is used to create a rectangular grid out of two given
# one-dimensional arrays representing the Cartesian indexing or Matrix indexing.

ar1=np.array([1,2,3,4])
ar2=np.array([1,2,3,4])
print(np.meshgrid(ar1,ar2))
```

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

```
# 56 Element wise division

ar1=np.array([1,2,3,4])
ar2=np.array([1,2,2,2])
print(np.divide(ar1,ar2))
```

```
[1.  1.  1.5 2. ]
```

```
#57 np.unique function

# With the help of np.unique() method, we can get the unique values from an array
# given as parameter in np.unique() method.

a=np.unique([1, 1, 2, 2, 3, 3])
print(a)
```

```
[1 2 3]
```

```
#58 correlation coefficient of two numpy arrays

ar1=np.array([1,2,3,4])
ar2=np.array([1,2,2,2])
print(np.corrcoef(ar1, ar2) )
```

```
[[1.         0.77459667]
 [0.77459667 1.        ]]
```

```
#59 Use of np.pad function

# numpy.pad() function is used to pad the Numpy arrays. Sometimes there is a need
#  to perform padding in Numpy arrays, then numPy.pad() function is used.
#  The function returns the padded array of rank equal to the given array and
#  the shape will increase according to pad_width.

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


pad_arr = np.pad(arr, (3, 2), 'linear_ramp',
                 end_values=(-4, 5))

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

for i in arr:
  print(i)

ar = np.array([[1, 2, 3], [5, 6, 7], [9, 10, 11]])
for i in ar:
  for x in i:
     print(x)
```

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

```
# 60 Iterate over each element in a numpy array

a=np.array([1,2,3,4])
for x in a:               #for 1D array
  print(x)

b=np.array([[1,2,3],[4,5,6]])
for x in b:               #for 2D array
  for y in x:
      print(y)
```

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

```
# 61. Generate an array of 20 elements linearly spaced between 5 and 15.

var_arr = np.linspace(5, 15, 20)
print(var_arr)
```

```
[ 5.          5.52631579  6.05263158  6.57894737  7.10526316  7.63157895
  8.15789474  8.68421053  9.21052632  9.73684211 10.26315789 10.78947368
 11.31578947 11.84210526 12.36842105 12.89473684 13.42105263 13.94736842
 14.47368421 15.        ]
```

```
# 62. Write a function to compute the Manhattan distance between two 1D arrays,
# array1 and array2, of equal length.

def manhattan_distance(array1, array2):
    return np.sum(np.abs(array1 - array2))
a=np.array([1,2,3,4])
b=np.array([6,7,8,9])

x= manhattan_distance(a,b)
print(x)
```

```
20
```

```
# 63. Flatten a given 3x3 2D array into a 1D array.

array_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
array_1d = array_2d.flatten()
print(array_1d)
```

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

```
# 64. Write a script to create an array and then check if it contains any NaN values.

array = np.array([1, 2, np.nan, 4, 5])
new_nan = np.isnan(array).any()
print(new_nan)
```

```
True
```

```
# 65. Create a 5x5 identity matrix using NumPy.

identity_matrix = np.eye(5)
print(identity_matrix)
```

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

```
# 66. Determine the rank of a given 4x4 matrix.

matrix = np.array([[1,4,-1,1],[2,4,-2,2],[3,6,-6,3],[4,8,-4,4]])
rank = np.linalg.matrix_rank(matrix)
print(matrix)
print("Rank:",rank)
```

```
[[ 1  4 -1  1]
 [ 2  4 -2  2]
 [ 3  6 -6  3]
 [ 4  8 -4  4]]
Rank: 3
```

```
# 67. Sort the elements of a given 1D array in descending order.

array = np.array([6,3,7,9,3,5,2,1,8])
array_sorted = np.sort(array)

print(array_sorted)
```

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

```
# 68. Slice a 3x3x3 array to extract a 2x2x2 sub-array from its corner.

array_3d = np.array([[[3,4,5],[1,2,3],[2,3,4]],[[2,3,4],[6,7,8],[4,5,6]],[[5,6,7],[9,8,7]
sub_array = array_3d[:2, :2, :2]

print(sub_array)
```

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

 [[2 3]
  [6 7]]]
```

```
# 69. Calculate the median of a given 1D array without using NumPy’s median function.

array = np.array([7,3,4,7,9,1,2,6,9])
array_sorted = np.sort(array)

n = len(array_sorted)
if n % 2 == 0:
    median = (array_sorted[n//2-1] + array_sorted[n//2]) / 2
else:
    median = array_sorted[n//2]

print("Median is:",median)
```

```
 Median is: 6
```

```
# 70. Concatenate two given 2D arrays of the same shape along the second axis.
```