DATA SCIENCE & MACHINE LEARNING-LAB CYCLE 2

1. Create a three dimensional array specifying float data type and print it.

   CODE:

```
import numpy as np
a3d_array=np.array([[[1.1,2.1],[3.1,4.1]],[[5.1,6.1],[7.1,8.1]]],dtype=float)
print(a3d_array)
```

   OUTPUT:

```
In [10]: runfile('/home/sjcet/.config/spyder-py3/1-3d_array.py', wdir='/home/sjcet/.config/spyder-py3')
[[[1.1 2.1]
  [3.1 4.1]]

 [[5.1 6.1]
  [7.1 8.1]]]
```

2. Create a 2 dimensional array (2X3) with elements belonging to complex data type and print it. Also display
   a. the no: of rows and columns
   b. dimension of an array
   c. reshape the same array to 3X2

   CODE:

```
import numpy as np
x=np.array([[1+2j,1+3j,2+3j],[5+6j,3+8j,4+5j]])
print(x)
print("Number of Rows and Columns:",x.shape)
print("Reshaped matrix is:")
print(x.reshape(3,2))
print("Dimensions:",x.ndim)
```

   OUTPUT:

```
In [7]: runfile('/home/sjcet/.config/spyder-py3/2-2d_array.py', wdir='/home/sjcet/.config/spyder-py3')
[[1.+2.j 1.+3.j 2.+3.j]
 [5.+6.j 3.+8.j 4.+5.j]]
Number of Rows and Columns: (2, 3)
Reshaped matrix is:
[[1.+2.j 1.+3.j]
 [2.+3.j 5.+6.j]
 [3.+8.j 4.+5.j]]
Dimensions: 2
```

3. Familiarize with the functions to create
   a) an uninitialized array

b) array with all elements as 1,
c) all elements as 0

CODE:

```
import numpy as np
x=np.empty([2, 2])
print(x)
y=np.full((2, 2), 1)
print(y)
z=np.full((2, 2), 0)
print(z)
```

OUTPUT:

```
In [9]: runfile('/home/sjcet/.config/spyder-py3/p3.py', wdir='/home/sjcet/.config/spyder-py3')
[[4.68480746e-310 8.24069422e-071]
 [1.77462177e+160 1.22221901e+165]]
[[1 1]
 [1 1]]
[[0 0]
 [0 0]]
```

4. Create an one dimensional array using **arange** function containing 10 elements.
   Display
   a. First 4 elements
   b. Last 6 elements
   c. Elements from index 2 to 7

CODE:

```
import numpy as np
a = np.arange(1, 11, 1)
print(a)
first_element = a[:4]
print(first_element)
first_element1 = a[5:]
print(first_element1)
first_element2 = a[1:7]
print(first_element2)
```

OUTPUT:

```
In [12]: runfile('/home/sjcet/.config/spyder-py3/p4.py', wdir='/home/sjcet/.config/spyder-py3')
[ 1  2  3  4  5  6  7  8  9 10]
[1 2 3 4]
[ 6  7  8  9 10]
[2 3 4 5 6 7]
```

5. Create an 1D array with **arange** containing first 15 even numbers as elements

a. Elements from index 2 to 8 with step 2(also demonstrate the same using slice function)
b. Last 3 elements of the array using negative index
c. Alternate elements of the array
d. Display the last 3 alternate elements

CODE:

```
import numpy as np

array_1d=np.arange(0,30,2)
print("array is:",array_1d)
print("Elements from index 2 to 8 with step 2",array_1d[2:8:2])
s=slice(2, 8, 2)
print("Elements from index 2 to 8 with step 2 using slice",array_1d[s])
print("Last 3 elements of the array using negative index",array_1d[-3:-1])
print("Alternate elements of the array",array_1d[::2])
print("Display the last 3 alternate elements",array_1d[-3:-1:2])
print("Display the elements from indices 4 to 10 in descending order(use-values)")
print(array_1d[10:4:-1])
```

OUTPUT:

```
In [15]: runfile('/home/sjcet/.config/spyder-py3/p5.py', wdir='/home/sjcet/.config/spyder-py3')
array is: [ 0  2  4  6  8 10 12 14 16 18 20 22 24 26 28]
Elements from index 2 to 8 with step 2 [ 4  8 12]
Elements from index 2 to 8 with step 2 using slice [ 4  8 12]
Last 3 elements of the array using negative index [24 26]
Alternate elements of the array [ 0  4  8 12 16 20 24 28]
Display the last 3 alternate elements [24]
Display the elements from indices 4 to 10 in descending order(use-values)
[20 18 16 14 12 10]
```

6. Create a 2 Dimensional array with 4 rows and 4 columns.
   a. Display all elements excluding the first row
   b. Display all elements excluding the last column
   c. Display the elements of 1st and 2nd column in 2nd and 3rd row
   d. Display the elements of 2nd and 3rd column
   e. Display 2nd and 3rd element of 1st row
   f. Display the elements from indices 4 to 10 in descending order(use –values)

CODE:

```
import numpy as np

array_2d=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
print(array_2d)

print("Display all elements excluding the first row")
print(array_2d[1:4,:])
print("Display all elements excluding the last column")
```

```
print(array_2d[:,0:3])
print("Display the elements of 1 st and 2 nd column in 2 nd and 3 rd row")
print(array_2d[1:3,0:2])
print("Display the elements of 2 nd and 3 rd column")
print(array_2d[:,1:3])
print("Display 2 nd and 3 rd element of 1 st row")
print(array_2d[0,1:3])
```

OUTPUT:

```
In [17]: runfile('/home/sjcet/.config/spyder-py3/p6.py', wdir='/home/sjcet/.config/spyder-py3')
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Display all elements excluding the first row
[[ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Display all elements excluding the last column
[[ 1  2  3]
 [ 5  6  7]
 [ 9 10 11]
 [13 14 15]]
Display the elements of 1 st and 2 nd column in 2 nd and 3 rd row
[[ 5  6]
 [ 9 10]]
Display the elements of 2 nd and 3 rd column
[[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
Display 2 nd and 3 rd element of 1 st row
[2 3]
```

7. Create two 2D arrays using array object and
    a. Add the 2 matrices and print it
    b. Subtract 2 matrices
    c. Multiply the individual elements of matrix
    d. Divide the elements of the matrices
    e. Perform matrix multiplication
    f. Display transpose of the matrix
    g. Sum of diagonal elements of a matrix

CODE:

```
import numpy as np

M1 = np.array([[3, 6], [14, 21]])
M2 = np.array([[9, 27], [11, 22]])

M3 = M1 + M2
print("Matrix addition")
print(M3)
```

```
M3 = M1 - M2
print("Matrix Substract")
print(M3)

M4 = np.array([[3, 6], [5, -10]])
M5 = np.array([[9, -18], [11, 22]])
M3 = M4 * M5
print("Multiply the individual elements of matrix")
print(M3)

M3 = M1 / M2
print("Divide the elements of the matrices")
print(M3)

M3 = M4.dot(M5)
print("matrix multiplication")
print(M3)

M6 = np.array([[3, 6, 9], [5, -10, 15], [4,8,12]])
M7 = M6.transpose()
print("Transpose of the matrix")
print(M2)

print("Sum of diagonal elements of a matrix")
print(np.trace(M6))
```

OUTPUT:

```
In [4]: runfile('/home/sjcet/.config/spyder-py3/untitled1.py', wdir='/home/sjcet/.config/spyder-py3')
Matrix addition
[[12 33]
 [25 43]]
Matrix Substract
[[ -6 -21]
 [  3  -1]]
Multiply the individual elements of matrix
[[  27 -108]
 [  55 -220]]
Divide the elements of the matrices
[[0.33333333 0.22222222]
 [1.27272727 0.95454545]]
matrix multiplication
[[  93   78]
 [ -65 -310]]
Transpose of the matrix
[[ 9 27]
 [11 22]]
Sum of diagonal elements of a matrix
5
```

8. Demonstrate the use of insert() function in 1D and 2D array

CODE:

```
import numpy as np

arr = np.arange(5)
print("1D arr : \n", arr)

a = np.insert(arr, 1, 9)
print("Array after inserting '9' : \n", a)

arr = np.arange(12).reshape(3, 4)
print("2D arr : \n", arr)

a = np.insert(arr, 1, 9, axis = 1)
print("Array after inserting '9' : \n", a)
```

OUTPUT:

```
In [10]: runfile('/home/sjcet/Desktop/dsml-VicterJohney/cycle2/untitled2.py', wdir='/home/sjcet/
Desktop/dsml-VicterJohney/cycle2')
1D arr :
 [0 1 2 3 4]
Array after inserting '9' :
 [0 9 1 2 3 4]
2D arr :
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
Array after inserting '9' :
 [[ 0  9  1  2  3]
 [ 4  9  5  6  7]
 [ 8  9  9 10 11]]
```

9. Demonstrate the use of diag() function in 1D and 2D array.

CODE:

```
import numpy as np

arr = np.arange(1,6).reshape(1,5)
print("1D arr : \n", arr)

a = np.diag(arr)
print("1D Array Diagonal values : \n", a)

arr = np.arange(12).reshape(4, 3)
print("2D arr : \n", arr)

a = np.diag(arr)
print("2D Array diagonal values : \n", a)
```

OUTPUT:

```
In [22]: runfile('/home/sjcet/Desktop/dsml-VicterJohney/cycle2/untitled3.py', wdir='/home/sjcet/Desktop/dsml-VicterJohney/cycle2')
1D arr :
 [[1 2 3 4 5]]
1D Array Diagonal values :
 [1]
2D arr :
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
2D Array diagonal values :
 [0 4 8]
```

10. Demonstarte the use of append() function in 1D and 2D array

CODE:

```
import numpy as np

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

a = np.append(arr, 88)
print("1D Array Diagonal values : \n", a)

arr = np.array( [ [1, 2, 3],[ 4, 5, 6] ])
print("2D arr : \n", arr)

a = np.append(arr, [22, 23, 24])
print("2D Array diagonal values : \n", a)
```

OUTPUT:

```
In [23]: runfile('/home/sjcet/Desktop/dsml-VicterJohney/cycle2/untitled4.py', wdir='/home/sjcet/Desktop/dsml-VicterJohney/cycle2')
1D Array Diagonal values :
 [ 1  2  3  4  5  6  7 88]
2D arr :
 [[1 2 3]
 [4 5 6]]
2D Array diagonal values :
 [ 1  2  3  4  5  6 22 23 24]
```

11. Demonstarte the use of sum() function in 1D and 2D array.

CODE:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])
print("1D arr : \n", arr)

a = np.sum(arr)
print("1D Array Sum : \n", a)

arr = np.array( [ [1, 2, 3],[ 4, 5, 6],[7, 8, 9] ])
print("2D arr : \n", arr)
```

```
a = np.sum(arr)
print("2D Array Sum : \n", a)
```

OUTPUT:

```
In [6]: runfile('/home/sjcet/Desktop/dsml-VicterJohney/cycle2/p11.py', wdir='/home/sjcet/Desktop/dsml-VicterJohney/cycle2')
1D arr :
 [1 2 3 4 5 6 7]
1D Array Sum :
 28
2D arr :
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
2D Array Sum :
 45
```

12. Create a square matrix with random integer values(use randint()) and use appropriate functions to find:

i) inverse
ii) rank of matrix
iii) Determinant
iv) transform matrix into 1D array
v) eigen values and vectors

CODE:

```
import numpy as np
from numpy import random as r

x = r.randint(100, size=(3, 3))
print ("Square matrix with random numbers: \n",x)

print("Inverse: \n", np.linalg.inv(x))

print("Matrix Rank: \n", np.linalg.matrix_rank(x))

print("Determinant: \n", np.linalg.det(x))

print("Transform matrix into 1D: \n", np.ravel(x))

print("Eigen Value: \n", np.linalg.eig(x))
```

OUTPUT:

```
In [31]: runfile('/home/sjcet/Desktop/dsml-VicterJohney/cycle2/untitled1.py', wdir='/home/sjcet/
Desktop/dsml-VicterJohney/cycle2')
Square matrix with random numbers:
 [[36 63 61]
 [37 83 99]
 [84 20 82]]
Inverse:
 [[ 0.03819549 -0.03123071  0.00929165]
 [ 0.04180451 -0.01719034 -0.01034428]
 [-0.04932331  0.0361852   0.00519984]]
Matrix Rank:
 3
Determinant:
 126350.00000000001
Transform matrix into 1D:
 [36 63 61 37 83 99 84 20 82]
Eigen Value:
 (array([186.90181365 +0.j        ,   7.04909317+25.02665954j,
         7.04909317-25.02665954j]), array([[-0.5002026 +0.j        ,  0.43998897-0.25712099j,
         0.43998897+0.25712099j],
       [-0.68402517+0.j        ,  0.5088864 +0.29294191j,
         0.5088864 -0.29294191j],
       [-0.53094908+0.j        , -0.62890235+0.j        ,
        -0.62890235-0.j        ]]))
```

13. Create a matrix X with suitable rows and columns
i) Display the cube of each element of the matrix using different methods
(use multiply(), *, power(),**)
ii) Display identity matrix of the given square matrix.
iii) Display each element of the matrix to different powers.
iv) Create a matrix Y with same dimension as X and perform the operation X^2 +2Y

CODE:

```
import numpy as np

x = np.arange(1,10).reshape(3,3)
print(x)

print("Matrix cube using multiply(): \n",np.multiply(x,(x*x)))

print("Matrix cube using pow()): \n",np.power(x,3))

print("Matrix cube using *: \n",x*x*x)

print("Matrix cube using **: \n",x**3)

print("Identity Matrix of 3x3: \n",np.identity(3,dtype=int))

print("each element of the matrix to different powers: \n",np.power(x,x))

y = np.arange(11,20).reshape(3,3)
#print("x : \n ",x)
#print("y : \n ",y)
#print("x^2 : \n ",np.power(x,2))
#print("2y : \n ",np.multiply(y,2))
```

print("perform the operation X^2 +2Y: \n",np.add((np.power(x,2)),(np.multiply(y,2))))

OUTPUT:

```
In [7]: runfile('/home/sjcet/Desktop/dsml-VicterJohney/cycle2/p13.py', wdir='/home/sjcet/Desktop/dsml-VicterJohney/cycle2')
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Matrix cube using multiply():
 [[  1   8  27]
 [ 64 125 216]
 [343 512 729]]
Matrix cube using pow()):
 [[  1   8  27]
 [ 64 125 216]
 [343 512 729]]
Matrix cube using *:
 [[  1   8  27]
 [ 64 125 216]
 [343 512 729]]
Matrix cube using **:
 [[  1   8  27]
 [ 64 125 216]
 [343 512 729]]
Identity Matrix of 3x3:
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
each element of the matrix to different powers:
 [[        1         4        27]
 [      256      3125     46656]
 [   823543  16777216 387420489]]
perform the operation X^2 +2Y:
 [[ 23  28  35]
 [ 44  55  68]
 [ 83 100 119]]
```

14. Multiply a matrix with a submatrix of another matrix and replace the same in larger matrix.

CODE:

```
import numpy as np;

x = np.arange(1,31).reshape(5,6)
print("x-> big matrix :\n",x)
#x[column,row]
y=x[1:4,2:5]

#print("y-> sub matrix of x : \n",y)

z = np.arange(2,11).reshape(3,3)
#print("matrix z :\n",z)

y = np.multiply(y,z)

#print("multiplied matrix : \n",y)

x[1:4,2:5] = y

print("final replaced matrix:\n ",x)
```

OUTPUT:

```
In [25]: runfile('/home/sjcet/Desktop/dsml-VicterJohney/cycle2/p14.py', wdir='/home/sjcet/Desktop/
dsml-VicterJohney/cycle2')
matrix :
 [[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]
 [25 26 27 28 29 30]]
replaced(3x3) matrix:
  [[  1   2   3   4   5   6]
 [  7   8  18  30  44  12]
 [ 13  14  75  96 119  18]
 [ 19  20 168 198 230  24]
 [ 25  26  27  28  29  30]]
```

15. Given 3 Matrices A, B and C. Write a program to perform matrix multiplication of the 3 matrices.

CODE:

```
import numpy as np

A =np.array( [[12, 7, 3],[4, 5, 6],[7, 8, 9]])
print("Matrix A:\n ",A)
B = np.array([[5, 8, 1, 2],[6, 7, 3, 0],[4, 5, 9, 1]])
print("Matrix B:\n ",B)
C = np.array([[1, 7, 3],[3, 5, 6],[6, 8, 9],[2, 9, 1]])
print("Matrix C:\n ",C)
result=np.dot(A,B)
#print(result)
new=np.dot(result,C)
print("Products of 3 mtrices is: \n", new)
#print(np.dot(np.dot(A,B),C))
```

OUTPUT:

```
In [15]: runfile('/home/sjcet/Desktop/dsml-VicterJohney/cycle2/untitled0.py', wdir='/home/sjcet/
Desktop/dsml-VicterJohney/cycle2')
Matrix A:
  [[12  7  3]
 [ 4  5  6]
 [ 7  8  9]]
Matrix B:
  [[5 8 1 2]
 [6 7 3 0]
 [4 5 9 1]]
Matrix C:
  [[1 7 3]
 [3 5 6]
 [6 8 9]
 [2 9 1]]
Products of 3 mtrices is:
 [[1008 2321 1869]
 [ 831 1713 1475]
 [1308 2721 2330]]
```

16. Write a program to check whether given matrix is symmetric or Skew Symmetric.

CODE:

```
import numpy as np;

x = np.arange(1,10).reshape(3,3)
print("Matrix x: \n", x)
# Transposing the Matrix M
y=x.transpose();
print("Transpose of x :\n ",y)
# -1(Transpose)
skew=np.multiply(y,-1);
print("negative of transpose :\n ",skew)
if x.all() == y.all():
    print("Matrix is symmetric")
else:
    print("Matrix is not symmetric")
if x.all() == ~y.all():
    print("Matrix is skew symmetric")
else:
    print("Matrix is not skew matrix")
```

OUTPUT:

```
In [81]: runfile('/home/sjcet/Desktop/dsml-VicterJohney/cycle2/p16.py', wdir='/home/sjcet/Desktop/
dsml-VicterJohney/cycle2')
Matrix x:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Transpose of x :
  [[1 4 7]
 [2 5 8]
 [3 6 9]]
negative of transpose :
  [[-1 -4 -7]
 [-2 -5 -8]
 [-3 -6 -9]]
Matrix is symmetric
Matrix is not skew matrix
```

17. Write a program to find out the value of X using solve(), given A and b as above

CODE:

```
import numpy as np;

A=np.array([[2,1,-2],[3,0,1],[1,1,-1]])
b=np.array([[-3],[5],[2]])

print("A:\n",A)
print("b:\n",b)

X=np.multiply((np.linalg.inv(A)),b)
print("X is : \n",X)
```

OUTPUT:

```
In [85]: runfile('/home/sjcet/Desktop/dsml-VicterJohney/cycle2/p17.py', wdir='/home/sjcet/Desktop/dsml-VicterJohney/cycle2')
A:
 [[ 2  1 -2]
 [ 3  0  1]
 [ 1  1 -1]]
b:
 [[-3]
 [ 5]
 [ 2]]
X is :
 [[-0.75 -0.75  0.75]
 [-5.    0.   10.  ]
 [-1.5   0.5   1.5 ]]
```

18. Write a program to perform the SVD of a given matrix. Also reconstruct the given matrix from the 3 matrices obtained after performing SVD.

CODE:

```
import numpy as np;

A = np.array([[1, 2], [3, 4], [5, 6]])
print("Matrix:\n",A)
# SVD
U, s, VT = np.linalg.svd(A)
print("U:\n", U)
print("Sigma:\n",s)
print("V^T:\n",VT)
```

OUTPUT:

```
In [89]: runfile('/home/sjcet/Desktop/dsml-VicterJohney/cycle2/untitled4.py', wdir='/home/sjcet/Desktop/dsml-VicterJohney/cycle2')
Matrix:
 [[1 2]
 [3 4]
 [5 6]]
U:
 [[-0.2298477   0.88346102  0.40824829]
 [-0.52474482  0.24078249 -0.81649658]
 [-0.81964194 -0.40189603  0.40824829]]
Sigma:
 [9.52551809 0.51430058]
V^T:
 [[-0.61962948 -0.78489445]
 [-0.78489445  0.61962948]]
```