Sixth Semester Computer Science and Engineering
Open Lab: Python for Machine Learning
Lab sheet 3 : Programming Assignments

In a Jupyter notebook use NumPy to solve the following:

1. convert a list of numeric value into a one-dimensional NumPy array. _
   Expected Output:
   ```
   Original List: [12.23, 13.32, 100, 36.32]
   One-dimensional numpy array: [ 12.23 13.32 100. 36.32]
   ```

2. Create a 3x3 matrix with values ranging from 2 to 10. _
   Expected Output:
   ```
   [[ 2 3 4]
   [ 5 6 7]
   [ 8 9 10]]
   ```

3. create a null vector of size 10 and update sixth value to 11._
   ```
   [ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
   Update sixth value to 11
   [ 0. 0. 0. 0. 0. 0. 11. 0. 0. 0.]
   ```

4. create a array with values ranging from 12 to 38._
   Expected Output:
   ```
   [12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37]
   ```

5. reverse an array (first element becomes last). _
   ```
   Original array:
   [12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37]
   Reverse array:
   [37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12]
   ```

6. create a 2d array with 1 on the border and 0 inside. _
   Expected Output:
   ```
   Original array:
   [[ 1. 1. 1. 1. 1.]
   [ 1. 1. 1. 1. 1.]
   [ 1. 1. 1. 1. 1.]
   [ 1. 1. 1. 1. 1.]
   [ 1. 1. 1. 1. 1.]]
   1 on the border and 0 inside in the array
   [[ 1. 1. 1. 1. 1.]
   [ 1. 0. 0. 0. 1.]
   [ 1. 0. 0. 0. 1.]
   [ 1. 0. 0. 0. 1.]
   [ 1. 1. 1. 1. 1.]]
   ```

7. create a 8x8 matrix and fill it with a checkerboard pattern.

   ```
   Checkerboard pattern:
   [[0 1 0 1 0 1 0 1]
   [1 0 1 0 1 0 1 0]
   [0 1 0 1 0 1 0 1]
   [1 0 1 0 1 0 1 0]
   [0 1 0 1 0 1 0 1]
   [1 0 1 0 1 0 1 0]
   [0 1 0 1 0 1 0 1]
   [1 0 1 0 1 0 1 0]]
   ```

8. convert a list and tuple into arrays.

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

9. find the number of elements of an array, length of one array element in bytes and total bytes consumed by the elements.

   ```
   Expected Output:
   Size of the array: 3
   Length of one array element in bytes: 8
   Total bytes consumed by the elements of the array: 24
   ```

10. find the set difference of two arrays. The set difference will return the sorted, unique values in array1 that are not in array2.

    ```
    Expected Output:
    Array1: [ 0 10 20 40 60 80]
    Array2: [10, 30, 40, 50, 70, 90]
    Set difference between two arrays:
    [ 0 20 60 80]
    ```

11. find the set exclusive-or of two arrays. Set exclusive-or will return the sorted, unique values that are in only one (not both) of the input arrays.

    ```
    Array1: [ 0 10 20 40 60 80]
    Array2: [10, 30, 40, 50, 70]
    Unique values that are in only one (not both) of the input arrays:
    [ 0 20 30 50 60 70 80]
    ```

12. find the union of two arrays. Union will return the unique, sorted array of values that are in either of the two input arrays.

    ```
    Array1: [ 0 10 20 40 60 80]
    Array2: [10, 30, 40, 50, 70]
    Unique sorted array of values that are in either of the two input arrays:
    [ 0 10 20 30 40 50 60 70 80]
    ```

13. construct an array by repeating.
14. Sample array: [1, 2, 3, 4]

    ```
    Expected Output:
    Original array
    ```

```
[1, 2, 3, 4]
Repeating 2 times
[1 2 3 4 1 2 3 4]
Repeating 3 times
[1 2 3 4 1 2 3 4 1 2 3 4]
```

15. compare two arrays using numpy.

```
Array a: [1 2]
Array b: [4 5]
a > b
[False False]
a >= b
[False False]
a < b
[ True True]
a <= b
[ True True]
```

16. create an array of ones and an array of zeros.

Expected Output:

```
Create an array of zeros
Default type is float
[[ 0. 0.]]
Type changes to int
[[0 0]]
Create an array of ones
Default type is float
[[ 1. 1.]]
Type changes to int
[[1 1]]
```

17. change the dimension of an array.

Expected Output:

```
6 rows and 0 columns
(6,)
(3, 3) -> 3 rows and 3 columns
[[1 2 3]
[4 5 6]
[7 8 9]]
Change array shape to (3, 3) -> 3 rows and 3 columns
[[1 2 3]
[4 5 6]
[7 8 9]]
```

18. create a contiguous flattened array.

```
Original array:
[[10 20 30]
[20 40 50]]
New flattened array:
```

```
[10 20 30 20 40 50]
```

19. create a 2-dimensional array of size 2 x 3 (composed of 4-byte integer elements), also print the shape, type and data type of the array.
    Expected Output:

    ```
    (2, 3)
    int32
    ```

20. create a new shape to an array without changing its data.

    ```
    Reshape 3x2:
    [[1 2]
    [3 4]
    [5 6]]
    Reshape 2x3:
    [[1 2 3]
    [4 5 6]]
    ```

21. change the data type of an array.
    Expected Output:

    ```
    [[ 2 4 6]
    [ 6 8 10]]
    Data type of the array x is: int32
    New Type: float64
    [[ 2. 4. 6.]
    [ 6. 8. 10.]]
    ```

22. create a new array of 3*5, filled with 2.
    Expected Output:

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

23. create an array of 10's with the same shape and type of an given array.
    Sample array: x = np.arange(4, dtype=np.int64)
    Expected Output:

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

24. create a 3-D array with ones on the diagonal and zeros elsewhere.
    Expected Output:

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

25. create a 2-D array whose diagonal equals [4, 5, 6, 8] and 0's elsewhere.
    Expected Output:

    ```
    [[4 0 0 0]
    [0 5 0 0]
    ```

```
[0 0 6 0]
[0 0 0 8]]
```

26. create a 1-D array going from 0 to 50 and an array from 10 to 50.
    Expected Output:
    Array from 0 to 50:
    [0 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 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]
    Array from 10 to 50:
    [10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
    35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]

27. find the 4th element of a specified array.
    Expected Output:
    [[2 4 6]
    [6 8 10]]
    Forth e1ement of the array:
    6

28. interchange two axes of an array.
    Sample array: [[1 2 3]]
    Expected Output:
    [[1]
    [2]
    [3]]

29. insert a new axis within a 2-D array.
    2-D array of shape (3, 4).
    Expected Output:
    New shape will be will be (3, 1, 4).

30. concatenate two 2-dimensional arrays.
    Expected Output:
    Sample arrays: ([[0, 1, 3], [5, 7, 9]], [[0, 2, 4], [6, 8, 10]]
    Expected Output:
    [[0 1 3 0 2 4]
    [5 7 9 6 8 10]]

31. split an array of 14 elements into 3 arrays, each of which has 2, 4, and 8 elements in the original order.
    Expected Output:
    Original array: [1 2 3 4 5 6 7 8 9 10 11 12 13 14]
    After splitting:
    [array([1, 2]), array([3, 4, 5, 6]), array([7, 8, 9, 10, 11, 12, 13, 14])]

32. split an of array of shape 4x4 it into two arrays along the second axis.
    Sample array :
    [[0 1 2 3]
    [4 5 6 7]
    [8 9 10 11]
    [12 13 14 15]]

Expected Output:

```
[array([[ 0, 1],
[ 4, 5],
[ 8, 9],
[12, 13]]), array([[ 2, 3],
[ 6, 7],
[10, 11],
[14, 15]]), array([], shape=(4, 0), dtype=int64)]
```

33. **64.** create a 5x5 matrix with row values ranging from 0 to 4.

    Original array:

    ```
    [[ 0. 0. 0. 0. 0.]
    [ 0. 0. 0. 0. 0.]
    [ 0. 0. 0. 0. 0.]
    [ 0. 0. 0. 0. 0.]
    [ 0. 0. 0. 0. 0.]]
    Row values ranging from 0 to 4.
    [[ 0. 1. 2. 3. 4.]
    [ 0. 1. 2. 3. 4.]
    [ 0. 1. 2. 3. 4.]
    [ 0. 1. 2. 3. 4.]
    [ 0. 1. 2. 3. 4.]]
    ```

34. create a vector of size 10 with values ranging from 0 to 1, both excluded.

    Expected Output:

    ```
    [ 0.09090909 0.18181818 0.27272727 0.36363636 0.45454545 0.54545455
    0.63636364 0.72727273 0.81818182 0.90909091]
    ```

35. create an array with 10^3 elements.

    Expected Output:

    ```
    [ 0. 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. 31. 32. 33. 34. 35.
    - - - - - - - - - - - - - - - - - - - -
    972. 973. 974. 975. 976. 977. 978. 979. 980. 981. 982. 983.
    984. 985. 986. 987. 988. 989. 990. 991. 992. 993. 994. 995.
    996. 997. 998. 999.]
    ```

36. remove specific elements in a numpy array.

    Expected Output:

    ```
    Original array:
    [ 10 20 30 40 50 60 70 80 90 100]
    ```

```
Delete first, fourth and fifth elements:
[ 20 30 60 70 80 90 100]
```

37. remove the negative values in a numpy array with 0.

Expected Output:

```
Original array:
[-1 -4 0 2 3 4 5 -6]
Replace the negative values of the said array with 0:
[0 0 0 2 3 4 5 0]
```

38. remove all rows in a numpy array that contain non-numeric values.

Expected Output:

```
Original array:
[[ 1. 2. 3.]
[ 4. 5. nan]
[ 7. 8. 9.]
[ 1. 0. 1.]]
Remove all non-numeric elements of the said array
[[ 1. 2. 3.]
[ 7. 8. 9.]
[ 1. 0. 1.]]
```

39. remove all rows in a numpy array that contain non-numeric values.

Sample array :

```
a = np.array([97, 101, 105, 111, 117])
b = np.array(['a','e','i','o','u'])
```

Note: Select the elements from the second array corresponding to elements in the first array that are greater than 100 and less than 110

Expected Output:

```
Original arrays
[ 97 101 105 111 117]
['a' 'e' 'i' 'o' 'u']
Elements from the second array corresponding to elements in the first
array that are greater than 100 and less than 110:
['e' 'i']
```

40. check whether the numpy array is empty or not.

Expected Output:

```
2
0
```

41. divide each row by a vector element.

Expected Output:

```
Original array:
[[20 20 20]
[30 30 30]
[40 40 40]]
Vector:
[20 30 40]
[[ 1. 1. 1.]
[ 1. 1. 1.]
[ 1. 1. 1.]]
```

42. add, subtract, multiply, divide arguments element-wise.
    Expected Output:
    ```
    Add:
    5.0
    Subtract:
    -3.0
    Multiply:
    4.0
    Divide:
    0.25
    ```

43. get the element-wise remainder of an array of division.
    Sample Output:
    ```
    Original array:
    [0 1 2 3 4 5 6]
    Element-wise remainder of division:
    [0 1 2 3 4 0 1]
    ```

44. round array elements to the given number of decimals.
    Sample Output:
    ```
    [ 1. 2. 2.]
    [ 0.3 0.5 0.6]
    [ 0. 2. 2. 4. 4.]
    ```

45. multiply a 5x3 matrix by a 3x2 matrix and create a real matrix product.
    Sample output:
    ```
    First array:
    [[ 0.44349753 0.81043761 0.00771825]
    [ 0.64004088 0.86774612 0.19944667]
    [ 0.61520091 0.24796788 0.93798297]
    [ 0.22156999 0.61318856 0.82348994]
    [ 0.91324026 0.13411297 0.00622696]]
    Second array:
    [[ 0.73873542 0.06448186]
    [ 0.90974982 0.06409165]
    [ 0.22321268 0.39147412]]
    Dot product of two arrays:
    [[ 1.06664562 0.08356133]
    [ 1.30677176 0.17496452]
    [ 0.88942914 0.42275803]
    [ 0.90534318 0.37596252]
    [ 0.79804212 0.06992065]]
    ```

46. create an inner product of two arrays.
    Sample Output:

```
Array x:
[[[ 0 1 2 3]
[ 4 5 6 7]
[ 8 9 10 11]]
[[12 13 14 15]
[16 17 18 19]
[20 21 22 23]]]
Array y:
[0 1 2 3]
Inner of x and y arrays:
[[ 14 38 62]
[ 86 110 134]]
```

47. generate inner, outer, and cross products of matrices and vectors.
    Expected Output:

```
Matrices and vectors.
x:
[ 1. 4. 0.]
y:
[ 2. 2. 1.]
Inner product of x and y:
10.0
Outer product of x and y:
[[ 2. 2. 1.]
[ 8. 8. 4.]
[ 0. 0. 0.]]
Cross product of x and y:
[ 4. -1. -6.]
```

48. generate a matrix product of two arrays.
    Sample Output:

```
Matrices and vectors.
x:
[[1, 0], [1, 1]]
y:
[[3, 1], [2, 2]]
Matrix product of above two arrays:
[[3 1]
[5 3]]
```

49. calculate mean across dimension, in a 2D numpy array.
    Sample output:

```
Original array:
[[10 30]
[20 60]]
Mean of each row:
[ 15. 45.]
```

```
Mean of each column:
[ 20. 40.]
```

50. create a random array with 1000 elements and compute the average, variance, standard deviation of the array elements.

    Sample output:

    ```
    Average of the array elements:
    -0.0255137240796
    Standard deviation of the array elements:
    0.984398282476
    Variance of the array elements:
    0.969039978542
    ```

51. generate five random numbers from the normal distribution.

    Expected Output:

    ```
    [-0.43262625 -1.10836787 1.80791413 0.69287463 -0.53742101]
    ```

52. generate six random integers between 10 and 30.

    Expected Output:

    ```
    [20 28 27 17 28 29]
    ```

53. create a 3x3x3 array with random values.

    Expected Output:

    ```
    [[[ 0.48941799 0.58722213 0.43453926]
    [ 0.94497547 0.81081709 0.1510409 ]
    [ 0.66657127 0.29494755 0.48047144]]
    [[ 0.02287253 0.95232614 0.32264936]
    [ 0.67009741 0.25458304 0.16290913]
    [ 0.15520198 0.86826529 0.9679322 ]]
    [[ 0.13503103 0.02042211 0.24683897]
    [ 0.97852158 0.22374748 0.10798856]
    [ 0.62032646 0.5893892 0.16958144]]]
    ```

54. create a random 10x4 array and extract the first five rows of the array and store them into a variable.

    Sample Output:

    ```
    Original array:
    [[ 0.38593391 0.52823544 0.8994567 0.22097238]
    [ 0.16639229 0.74964167 0.58102198 0.2811601 ]
    [ 0.56447627 0.42575759 0.71297527 0.91099347]
    [ 0.00261548 0.0064798 0.66096109 0.54514293]
    [ 0.7216008 0.95815426 0.53370551 0.28116107]
    [ 0.16252081 0.26191659 0.40883164 0.60653848]
    [ 0.55934457 0.37814126 0.63287808 0.01856616]
    [ 0.03788236 0.22705078 0.82024426 0.83019741]
    [ 0.31140166 0.43926341 0.38685152 0.92402934]
    ```

```
[ 0.00581032 0.83925377 0.95246879 0.28570894]]
First 5 rows of the above array:
[[ 0.38593391 0.52823544 0.8994567 0.22097238]
[ 0.16639229 0.74964167 0.58102198 0.2811601 ]
[ 0.56447627 0.42575759 0.71297527 0.91099347]
[ 0.00261548 0.0064798 0.66096109 0.54514293]
[ 0.7216008 0.95815426 0.53370551 0.28116107]]
```

55. normalize a 3x3 random matrix.
    Sample output:
    ```
    Original Array:
    [[ 0.87311805 0.96651849 0.98078621]
    [ 0.26407141 0.46784012 0.69947627]
    [ 0.20013296 0.75510414 0.26290783]]
    After normalization:
    [[ 0.86207941 0.98172337 1. ]
    [ 0.08190378 0.34292711 0.63964803]
    [ 0. 0.71090613 0.08041325]]
    ```

56. create a random vector of size 10 and sort it.
    Expected Output:
    ```
    Original array:
    [ 0.73123073 0.67714015 0.95615347 0.4759837 0.88789818 0.6910404 2
    0.59996415 0.26144489 0.51618644 0.89943882]
    Sorted array:
    [ 0.26144489 0.4759837 0.51618644 0.59996415 0.67714015 0.6910404 2
    0.73123073 0.88789818 0.89943882 0.95615347]
    ```

57.

58. create random vector of size 15 and replace the maximum value by -1.
    Sample output:
    ```
    Original array:
    [ 0.34807512 0.76714463 0.40242311 0.5634299 0.84972926 0.92247789
    0.93791571 0.5127047 0.50796265 0.50074454 0.26067194 0.07207825
    0.04927934 0.95309433 0.14043974]
    Maximum value replaced by -1:
    [ 0.34807512 0.76714463 0.40242311 0.5634299 0.84972926 0.92247789
    0.93791571 0.5127047 0.50796265 0.50074454 0.26067194 0.07207825
    0.04927934 -1. 0.14043974]
    ```

59. find point by point distances of a random vector with shape (10,2) representing coordinates.
    Sample output:
    ```
    [[ 0. 0.09871078 0.42100075 0.75597269 0.52281832 0.13721998
    0.1761711 0.28689498 0.42061575 0.61315509]
    [ 0.09871078 0. 0.43978557 0.71086596 0.59696144 0.14701023
    0.26602812 0.19254215 0.36762701 0.68776127]
    ```

```
.....
[ 0.42061575 0.36762701 0.30691429 0.34395028 0.63326713 0.29974614
0.47787697 0.39922329 0. 0.70954707]
[ 0.61315509 0.68776127 0.4097455 0.85714768 0.09080178 0.55976699
```

60. convert cartesian coordinates to polar coordinates of a random 10x3 matrix representing cartesian coordinates.
    Expected Output:

    ```
    [ 0.89225122 0.68774813 0.20392039 1.22093243 1.24435921 1.00358852
    0.37378547 0.8534585 0.31999648 0.567451 ]
    [ 1.02751197 1.26964967 0.02567519 0.85386412 0.73152767 0.45822494
    1.50634505 1.47389983 0.80818521 0.33001182]
    ```