Numpy Operations

1. Create a 2D numpy array, for each of the following:
    a. array
    b. arange
    c. randn

```
# Using array
np_arr = np.array([*range(1, 5)])
print(np_arr)
#[1 2 3 4]

# Using arange
np_arr = np.arange(1, 5)
print(np_arr)
# [1 2 3 4]

# Using randn
np_arr = np.random.randn(2, 4)
print(np_arr)
# [[-0.46587348  0.12463593 -0.96983394  0.54144899]
# [ 1.13436859  0.99851646  0.94148497  0.0647182 ]]
```

2. Create a 3D numpy array, for each of the following:
    a. zeros
    b. ones
    c. eye

```
# Using zeros
np.zeros(shape=[3] * 3, dtype=int)
# 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, 0, 0]]])

# Using ones
np.zeros(shape=[3] * 3, dtype=int)
# 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, 1]]]])

# Using eye
np.eye(4, dtype=int)
# array([[1, 0, 0, 0],
#       [0, 1, 0, 0],
#        [0, 0, 1, 0],
#        [0, 0, 0, 1]])
```

3. Create two (3,2) numpy arrays and, using vectorization, perform element-wise addition, subtraction, division of these two arrays.

```
arr1 = np.array([*range(1,7)]).reshape(3,2)
arr2 = np.array([*range(1, 12, 2)]).reshape(3,2)

print("Addition:\n", arr1 + arr2)
# Addition:
# [[ 2  5]
# [ 8 11]
# [14 17]]

print("Substraction:\n", arr1 - arr2)
# Substraction:
# [[ 0 -1]
# [-2 -3]
# [-4 -5]]

print("Division:\n", arr1 / arr2)
# Division:
# [[1.         0.66666667]
# [0.6        0.57142857]
# [0.55555556 0.54545455]]
```

4. Create a (2, 6) numpy array and, using broadcasting, add, subtract, multiply each element using the scalar value of ‘5’.

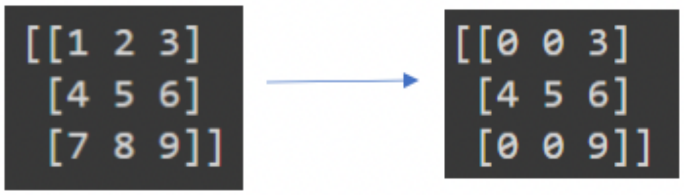
```
arr = np.array(range(1, 13)).reshape(2, 6)

print("Addition:\n", arr + 5)
# Addition:
# [[ 6  7  8  9 10 11]
# [12 13 14 15 16 17]]
```

```
print("Substract:\n", arr - 5)
# Substract:
# [[-4 -3 -2 -1  0  1]
# [ 2  3  4  5  6  7]]

print("Multiply:\n", arr * 5)
# Multiply:
# [[ 5 10 15 20 25 30]
# [35 40 45 50 55 60]]
```

5. Create a 1D numpy array having string values, and create a new array by type casting these string-values to their equivalent float values. (for example : "4.32" → 4.32)

```
arr = np.array([ f"{(num ** 2)/7}" for num in range(1,7)])

arrInt = arr.astype(np.float64)
print(arrInt)
# [0.14285714 0.57142857 1.28571429 2.28571429 3.57142857 5.14285714]
```

6. In the following array, replace the values '1, 2, 7, 8' with zeros, by writing one single line of



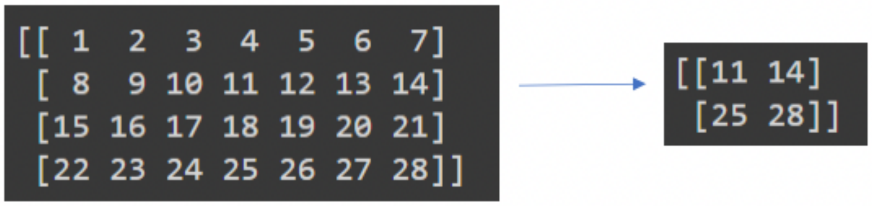
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
[[0 0 3]
 [4 5 6]
 [0 0 9]]
```

expression :

```
arr = np.array(range(1, 10)).reshape(3,3)

arr[ (arr == 1) | (arr == 2) | (arr == 7) | (arr==8) ] = 0

# or
# arr[np.isin(arr, (1,2, 7, 8))] = 0

print(arr)
# [[0 0 3]
# [4 5 6]
# [0 0 9]]
```

7. Given the following array of (4,7), using Boolean Indexing, display only these four values ‘11, 14, 25, 28’.

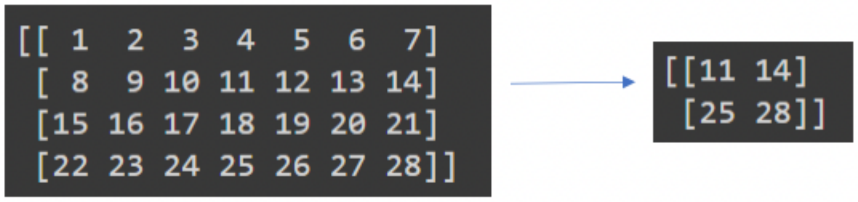


```
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14]
 [15 16 17 18 19 20 21]
 [22 23 24 25 26 27 28]]
```

```
[[11 14]
 [25 28]]
```

```
arr = np.array(range(1, 29)).reshape(4,7)
tmp_arr = arr[ :, [False, False, False, True, False, False, True]]
arr = tmp_arr[ [False, True, False, True], :]

print(arr)
# [[11 14]
# [25 28]]
```

8. Given the following array of (4,7), using Fancy Indexing, display only these four values ‘11, 14, 25, 28’.

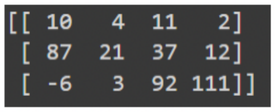


```
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14]
 [15 16 17 18 19 20 21]
 [22 23 24 25 26 27 28]]
```

```
[[11 14]
 [25 28]]
```

```
arr = np.array(range(1, 29)).reshape(4,7)

tmp_arr = arr[(1,3), :]
arr = tmp_arr[:, [3,6]]

print(arr)
# [[11 14]
# [25 28]]
```

9. Create the following array, and write expressions to find the mean, sum, square, squareroot, min, and max values for the following 2D array, across both the axes (0, 1).



```
[[ 10   4  11   2]
 [ 87  21  37  12]
 [ -6   3  92 111]]
```

```
arr = np.array([10, 4, 11, 2, 87, 21, 37, 12, -6, 3, 92,
111]).reshape(3,4)

print("mean: ", arr.mean()) # mean:  32.0

print("sum: ", arr.sum()) # sum:  384

print("squareroot: ", np.sqrt(abs(arr)))
#squareroot:  [[ 3.16227766  2.          3.31662479  1.41421356]
# [ 9.32737905  4.58257569  6.08276253  3.46410162]
# [ 2.44948974  1.73205081  9.59166305 10.53565375]]

print("min(axis0): ", arr.min(axis=0)) # min(axis0):  [-6  3 11  2]

print("min(axis1): ", arr.min(axis=1)) # min(axis1):  [ 2 12 -6]

print("max(axis0): ", arr.max(axis=0)) # max(axis0):  [ 87  21  92
111]

print("max(axis1): ", arr.max(axis=1)) # max(axis1):  [ 11  87 111]
```

10. Create two arrays using random method, and find the maximum, minimum values between them by using vectorization's element wise operations.

```
arr1 =  abs(np.random.randn(2, 4) * 10)
arr2 = abs(np.random.randn(2, 4) * 10)

print("minimum: ", np.minimum(arr1, arr2))
# minimum:  [[7.46030551 9.7639684  1.29711437 1.49255412]
# [1.51438003 0.80618224 5.52787063 2.91875281]]

print("max: ", np.maximum(arr1, arr2))
# max:  [[15.81077738 14.52468064 11.0459675  16.9272459 ]
# [12.24684768 15.62787255 19.97414017  9.32655722]]
```

11. Create two 1D arrays of length 10. Perform unique, intersection, union, set-difference operations on these arrays.

```
arr1 = np.array(["Orange", "Apple", "Pineapple", "Orange", "Mango",
"Apple"])

arr2 = np.array(["Tomato", "Mango", "Strawberry", "Banana",
"Pineapple"])

print("Unique: ", np.unique(arr1))
# Unique:  ['Apple' 'Mango' 'Orange' 'Pineapple']

print("Intersection: ", np.intersect1d(arr1, arr2))
# Intersection:  ['Mango' 'Pineapple']

print("Union: ", np.union1d(arr1, arr2))
# Union:  ['Apple' 'Banana' 'Mango' 'Orange' 'Pineapple'
# 'Strawberry' 'Tomato']

print("Set-difference: ", np.setdiff1d(arr1, arr2))
# Set-difference:  ['Apple' 'Orange']
```