Assignment 6

In [4]:

```
import numpy as np
```

Set 1:

Create a 3×3 array of all True Boolean values

In [5]:

```
arr=np.full_like(np.arange(1,10).reshape(3,3),True, dtype=bool)
arr
```

Out[5]:

```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

Extract odd numbers from an array

In [6]:

```
arr=np.arange(1,13)
odd_mask=(arr%2==1)
odd_nums=arr[odd_mask]
odd_nums
```

Out[6]:

```
array([ 1,  3,  5,  7,  9, 11])
```

Replace all odd numbers with -1

In [7]:

```
arr=np.arange(1,13)
odd_mask=(arr%2==1)
arr[odd_mask]=-1
arr
```

Out[7]:

```
array([-1,  2, -1,  4, -1,  6, -1,  8, -1, 10, -1, 12])
```

Reverse columns of a 2D array

In [8]:

```
print('original array:')
arr=np.arange(1,9).reshape(2,4)
print(arr)
arr=arr[:,::-1]
print('reversed array:')
print(arr)
```

```
original array:
[[1 2 3 4]
 [5 6 7 8]]
reversed array:
[[4 3 2 1]
 [8 7 6 5]]
```

Get indices of elements greater than 5 [hint: check where method of numpy]

In [9]:

```
arr=np.arange(1,8)
a1=np.where(arr>3)
print(a1)
```

```
(array([3, 4, 5, 6]),)
```

Access last row - last column

In [10]:

```
arr=np.arange(1,13).reshape(3,4)
print(arr)
print('\nlast-row-last-column:\n',arr[-1,-1])
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]

last-row-last-column:
 12
```

Set 2:

Normalize a NumPy array [to normalize means scaling the values of an array into a specific range, often [0, 1]]

In [11]:

```
arr=np.arange(1,13)
arr=np.sqrt(arr).reshape(3,4)
print(arr)
print('--------------')
varr=np.clip(arr,0,1)
print(varr)
```

```
[[1.         1.41421356 1.73205081 2.        ]
 [2.23606798 2.44948974 2.64575131 2.82842712]
 [3.         3.16227766 3.31662479 3.46410162]]
--------------
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

Create a 5×5 array with 1 on border and 0 inside

In [12]:

```
a1=np.ones(25).reshape(5,5)
a1
```

Out[12]:

```
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.]])
```

In [13]:

```
a2=np.zeros(9).reshape(3,3)
a2
```

```
Out[13]: array([[0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.]])
```

```
In [14]: a1[1:-1,1:-1]=a2
         a1
```

```
Out[14]: 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.]])
```

Create a random array of 10×10: perform stats (min, max, mean, std)

```
In [15]: a1=np.random.random(100).reshape(10,10)
         a1
```

```
Out[15]: array([[0.61865502, 0.41819319, 0.46755415, 0.19680715, 0.31566959,
                 0.2264288 , 0.03150167, 0.69048396, 0.04891835, 0.11154304],
                [0.84495566, 0.1443643 , 0.39321741, 0.04188511, 0.01093338,
                 0.87342983, 0.86298657, 0.22992247, 0.97826353, 0.08445621],
                [0.38491579, 0.73344023, 0.27699519, 0.31445709, 0.97233761,
                 0.2991543 , 0.55143418, 0.57557552, 0.84108644, 0.59043962],
                [0.26552264, 0.56232378, 0.34713353, 0.69097881, 0.47249472,
                 0.03205796, 0.79095841, 0.42796883, 0.00385187, 0.46387808],
                [0.80169658, 0.0327075 , 0.16291459, 0.95688928, 0.87835078,
                 0.38643876, 0.82884438, 0.74906802, 0.12591879, 0.84307361],
                [0.24293944, 0.70562079, 0.84661975, 0.71777983, 0.39281564,
                 0.45474814, 0.16437606, 0.79113066, 0.85740201, 0.60600181],
                [0.58467448, 0.61742232, 0.19216345, 0.53024265, 0.9890612 ,
                 0.18287888, 0.51075449, 0.72327177, 0.05455507, 0.98866204],
                [0.47563228, 0.6786637 , 0.61335974, 0.95217237, 0.68792981,
                 0.50071927, 0.24949755, 0.18120235, 0.57748485, 0.1039075 ],
                [0.69208257, 0.98109913, 0.22765375, 0.01781683, 0.41194704,
                 0.597998  , 0.45413834, 0.85343021, 0.34924055, 0.93506664],
                [0.71268701, 0.65816804, 0.44575601, 0.09090437, 0.77067098,
                 0.10463245, 0.57427546, 0.8229916 , 0.3665874 , 0.71909003]])
```

```
In [16]: print(np.min(a1))
```

```
0.0038518654145833775
```

```
In [17]: print(np.max(a1))
```

```
0.9890611963336609
```

```
In [18]: print(np.mean(a1))
```

```
0.49913002592451183
```

```
In [19]: print(np.std(a1))
```

```
0.29125092277057923
```

Broadcast a 1D array over a 2D array [all possible scenarios]

In [20]:

```
a1=np.arange(1,7)
a2=np.arange(11,23).reshape(2,6)
print(a1)
print('\n',a2)
```

```
[1 2 3 4 5 6]

 [[11 12 13 14 15 16]
 [17 18 19 20 21 22]]
```

In [21]:

```
print('Add\n',a1+a2)
```

```
Add
 [[12 14 16 18 20 22]
 [18 20 22 24 26 28]]
```

In [22]:

```
print('Subtract\n',a1-a2)
print('Subtract\n',a2-a1)
```

```
Subtract
 [[-10 -10 -10 -10 -10 -10]
 [-16 -16 -16 -16 -16 -16]]
Subtract
 [[10 10 10 10 10 10]
 [16 16 16 16 16 16]]
```

In [23]:

```
print('Multiply\n',a1*a2)
```

```
Multiply
 [[ 11  24  39  56  75  96]
 [ 17  36  57  80 105 132]]
```

In [24]:

```
print('Divide\n',a1/a2)
```

```
Divide
 [[0.09090909 0.16666667 0.23076923 0.28571429 0.33333333 0.375     ]
 [0.05882353 0.11111111 0.15789474 0.2        0.23809524 0.27272727]]
```

In [25]:

```
print('Int Divide\n',a2//a1)
```

```
Int Divide
 [[11  6  4  3  3  2]
 [17  9  6  5  4  3]]
```

In [26]:

```
print('Remainder\n',a1%a2)
```

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

In [27]:

```
age=np.random.randint(12,41,8)
age_grp=np.array(['Minor','Adult'])
print(age)
print(np.where(age<18,age_grp[0],age_grp[1]))
```

```
[27 36 18 23 29 28 23 31]
['Adult' 'Adult' 'Adult' 'Adult' 'Adult' 'Adult' 'Adult' 'Adult']
```

Sort a 2D array by column 1 [hint: check method argsort()]

In [28]:

```
# Approach 1
a1=np.random.randint(1,100,25).reshape(5,5)
print('array:\n',a1)
a1_indices=np.argsort(a1,0)
print('\nsorted by 1st column:')
print(a1[a1_indices,np.arange(a1.shape[1])])
```

```
array:
 [[56 95 60  4 41]
 [13 79 47 15 69]
 [38  5 98 88 64]
 [34 34 63 65 73]
 [ 1 48 94 10 59]]

sorted by 1st column:
[[ 1  5 47  4 41]
 [13 34 60 10 59]
 [34 48 63 15 64]
 [38 79 94 65 69]
 [56 95 98 88 73]]
```

In [29]:

```
# Approach 2
col1=a1[:,0]
index=np.argsort(col1)
print(a1[index])
```

```
[[ 1 48 94 10 59]
 [13 79 47 15 69]
 [34 34 63 65 73]
 [38  5 98 88 64]
 [56 95 60  4 41]]
```

Set 3:

Find common elements between two arrays [with and without using intersect1d]

In [30]:

```
a1=np.random.randint(1,52,12)
a2=np.random.randint(3,62,12)
print('With np.intersect1d():')
print(np.intersect1d(a1,a2))

# common_e=set(a1) & set(a2)
# print(common_e)
print('\nWithout np.intersect1d():')
print('(created mask using np.isin)')
print(a1[np.isin(a1,a2)])
```

```
With np.intersect1d():
[10 12]

Without np.intersect1d():
(created mask using np.isin)
[10 12]
```

Subtract row mean from each row of a 2D array

In [31]:

```
a1=np.arange(12,27).reshape(3,5)
print(a1)
mean_a=np.mean(a1,1)
mean_a
```

```
[[12 13 14 15 16]
 [17 18 19 20 21]
 [22 23 24 25 26]]
```

Out[31]:

```
array([14., 19., 24.])
```

Given a 4×4 array, swap the first half rows with the second half rows

In [32]:

```
a1=np.arange(1,17).reshape(4,4)
print(a1)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
```

Given a 1D array of random numbers, find the indices of the 5 largest values (no sorting/ check argsort() instead).

In [ ]:

```

```

From a 2D NumPy array, extract only the unique rows (no duplicates hint: check unique method).

In [ ]:

```

```

GeeksforGeeks

Power of 2

In [33]:

```
def isPowerofTwo(n):
        return n>0 and (n & (n-1)==0)
```

In [34]:

```
isPowerofTwo(1024)
```

Out[34]:

```
True
```

In [35]:

```
isPowerofTwo(1000)
```

Out[35]:

```
False
```

First and Last Occurrence

In [36]:

```
def find(arr, x):
        first_o=-1
        last_o=-1
        if x not in set(arr):
            return [-1,-1]
        for i in range(len(arr)):
            if arr[i]==x & first_o !=-1:
                print(i)
                first_o,last_o=i,i
            elif arr[i]==x:
                last_o=i
        return [first_o,last_o]
```

In [37]:

```
a=[6, 6, 6, 6, 7, 7, 7, 8]
find(a,8)
```

```
7
```

Out[37]:

```
[7, 7]
```

In [38]:

```
def is_power_of_two_bitwise(n):
    """Checks if n is a power of 2 using the bitwise AND operation."""
    # Must be positive (n > 0) AND have exactly one bit set to 1
    return n > 0 and (n & (n - 1) == 0)
```