ML LAB 01 (CS22104)

Q2:

2. Create an ndarray using the following data. Write the code and outputs.

In [29]:
```
data1 = np.array([1, 3, 9, 12, 18])
data2 = np.array([[1,5,2,0],[3,6,4,7]])
data3 = np.array([['7','6'],['5','4'],['3','2']])
data4 = np.array([[11,52,20,50],[34,61,43,17]])
```

In [31]:
```
print('data1')
print(data1)
print("data2")
print(data2)
print("data3")
print(data3)
print("data4")
print(data4)
```

```
data1
[ 1  3  9 12 18]
data2
[[1 5 2 0]
 [3 6 4 7]]
data3
[['7' '6']
 ['5' '4']
 ['3' '2']]
data4
[[11 52 20 50]
 [34 61 43 17]]
```

Q3:

3. Add Data1 in Data1. Multiply Data1 by 10. Convert Data1, Data2, Data3, and Data4 datatypes to integer.

In [6]:
```
addData1 = data1+data1
mulData1 = data1*10
```

In [7]:
```
print('Adding Data1 in Data1 = ',addData1)
print('Multiply Data1 by 10 = ',mulData1)
```

```
Adding Data1 in Data1 =  [ 2  6 18 24 36]
Multiply Data1 by 10 =  [ 10  30  90 120 180]
```

In [8]:
```
ConvertedData1 = data1.astype(np.int32)
ConvertedData2 = data2.astype(np.int32)
ConvertedData3 = data3.astype(np.int32)
ConvertedData4 = data4.astype(np.int32)
```

In [9]:
```
print(ConvertedData1.dtype)
print(ConvertedData2.dtype)
print(ConvertedData3.dtype)
print(ConvertedData4.dtype)
```

```
int32
int32
int32
int32
```

Q4:

4. Create a 3x5 array of 0s. 2x8 array of 1s, 6x3 array of integer random numbers

In [10]:
```
from numpy import random
```

In [11]:
```
array0s = np.full((3, 5), 0)
array0s
```

Out[11]:
```
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
```

In [12]:
```
array1s = np.full((2, 8), 1)
array1s
```

Out[12]:
```
array([[1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1]])
```

In [13]:
```
arrayRandomNumbers = random.randint(100,size=(6,3))
arrayRandomNumbers
```

Out[13]:
```
array([[73, 54, 63],
       [98, 16, 85],
       [47,  3, 74],
       [56, 60, 23],
       [87, 82, 56],
       [43, 37, 97]])
```

Q5:

5. Square the bold elements of Data1 and Data2. Hint: use indexing and slicing.

In [14]:
```
newData1 = data1.copy()
newData1[2:4] = newData1[2:4] **2
newData1
```

Out[14]:
```
array([  1,   3,  81, 144,  18])
```

In [15]:
```
newData2 = data2.copy()
newData2[0,1] = newData2[0,1] **2
newData2[1,2] = newData2[1,2] **2
newData2
```

Out[15]:
```
array([[ 1, 25,  2,  0],
       [ 3,  6, 16,  7]])
```

Q6:

6. Consider an array Subjects = [[$ML\ OS\ CCN$],[$OS\ AI\ MP$], [$CCN\ ML\ AI$]] Set all courses to 0 except ML and AI. Hint: use Boolean indexing.

In [16]:
```
Subjects = np.array([["ML","OS","CCN"],["OS","AI","MP"],["CCN","ML","AI"]])
# numpy built-in method
# newSubjects = np.isin(Subjects,["ML","AI"],invert=True)
# Subjects[newSubjects] = 0
# Subjects
# Boolean Indexing Method
Subjects[(Subjects != "ML") & (Subjects != "AI")] = 0
Subjects
```

Out[16]:
```
array([['ML', '0', '0'],
       ['0', 'AI', '0'],
       ['0', 'ML', 'AI']], dtype='<U3')
```

Q7:

7. Transpose the Data2, Data3, and Data4 arrays using appropriate function.

In [17]:
```
transposeData1 = data1.copy()
transposeData2 = data2.copy()
transposeData3 = data3.copy()
transposeData4 = data4.copy()
transposeData1 = np.array([transposeData1]).T
transposeData2 = transposeData2.T
transposeData3 = transposeData3.T
transposeData4 = transposeData4.T
print("Transpose Data1")
print(transposeData1)
print("Transpose Data2")
print(transposeData2)
print("Transpose Data3")
print(transposeData3)
print("Transpose Data4")
print(transposeData4)
```

```
Transpose Data1
[[ 1]
 [ 3]
 [ 9]
 [12]
 [18]]
Transpose Data2
[[1 3]
 [5 6]
 [2 4]
 [0 7]]
Transpose Data3
[['7' '5' '3']
 ['6' '4' '2']]
Transpose Data4
[[11 34]
 [52 61]
 [20 43]
 [50 17]]
```

Q8:

8. Concatenate Data2 and Data4 using vstack and hstack.

In [18]:
```
# Using Vstack
concat = np.vstack((data2,data4))
concat
```

Out[18]:
```
array([[ 1,  5,  2,  0],
       [ 3,  6,  4,  7],
       [11, 52, 20, 50],
       [34, 61, 43, 17]])
```

In [19]:
```
# Using hstack
concat = np.hstack((data2,data4))
concat
```

Out[19]:
```
array([[ 1,  5,  2,  0, 11, 52, 20, 50],
       [ 3,  6,  4,  7, 34, 61, 43, 17]])
```

Q9 and Q10:

9. Demean each column of Data2. Hint: use broadcasting.

In [20]:
```
meanData2 = data2.mean(0)
demeanData2 = data2 - meanData2
demeanData2
```

Out[20]:
```
array([[-1. , -0.5, -1. , -3.5],
       [ 1. ,  0.5,  1. ,  3.5]])
```

10. Convert Data2 shape into 4x2. Hint: Use reshape function.

In [21]:
```
newShapeData2 = data2.reshape(4,2)
newShapeData2
```

Out[21]:
```
array([[1, 5],
       [2, 0],
       [3, 6],
       [4, 7]])
```

Q11:

11. Consider the table having marks of three students scored in three subjects. Compute the maximum score of each subject. Hint: use max function ¶

In [22]:
```
subjectMarks = np.array([[60,70,50],[40,60,80],[70,55,65]])
print("Subject1 max Score",subjectMarks[0].max())
print("Subject2 max Score",subjectMarks[1].max())
print("Subject3 max Score",subjectMarks[2].max())
```

```
Subject1 max Score 70
Subject2 max Score 80
Subject3 max Score 70
```

Q12:

12. Compute the CGPA of all three students using the following formula and tables 1.11 and 1.12 (Hint:Use ufunc methods).

In [23]:
```
creditHours = np.array([2, 3, 4])
totalCreditHours = sum(creditHours)
def marks_to_grade_point(marks):
    if marks >= 85:
        return 4.0
    elif 80<=marks<= 84:
        return 3.7
    elif 75<=marks<= 79:
        return 3.4
    elif 70<=marks<= 74:
        return 3.0
    elif 67<=marks<= 69:
        return 2.7
    elif 64<=marks<= 66:
        return 2.4
    elif 60<=marks<= 63:
        return 2.0
    elif 57<=marks<= 59:
        return 1.7
    elif 54<=marks<= 56:
        return 1.4
    elif 50<=marks<= 53:
        return 1.0
    elif marks<50:
        return 0.0
CGPA = []
for student_id in range(subjectMarks.shape[1]):  # Loop through each student
    total_weighted_points = 0  # Initialize weighted points for the student
    for subjectID in range(subjectMarks.shape[0]):  # Loop through each subject
        marks = subjectMarks[subjectID][student_id]  # Marks for the student in this subject
```

```
        return 0.0
CGPA = []
for student_id in range(subjectMarks.shape[1]):  # Loop through each student
    total_weighted_points = 0  # Initialize weighted points for the student
    for subjectID in range(subjectMarks.shape[0]):  # Loop through each subject
        marks = subjectMarks[subjectID][student_id]  # Marks for the student in this subject
        gradePoint = marks_to_grade_point(marks)  # Convert marks to grade points
        total_weighted_points += gradePoint * creditHours[subjectID]  # Add weighted points
    # Calculating CGPA for the student
    studentcgpa = total_weighted_points / totalCreditHours
    CGPA.append(studentcgpa)

# Print the CGPA for each student
for i, studentcgpa in enumerate(CGPA, start=1):
    print(f"CGPA of Student {i}: {studentcgpa:.2f}")
```

```
CGPA of Student 1: 1.78
CGPA of Student 2: 1.96
CGPA of Student 3: 2.52
```

Q13 and Q14:

13. Compute the sorted unique values in subjects array.

In [24]:
```
Subjects = np.array([["ML","OS","CCN"],["OS","AI","MP"],["CCN","ML","AI"]])
uniqueSubjects = np.unique(Subjects)
uniqueSubjects
```

Out[24]:
```
array(['AI', 'CCN', 'ML', 'MP', 'OS'], dtype='<U3')
```

14. Compute the dot product and inverse of 4x4 random matrix. Hint: Use numpy.linalg functions.

In [25]:
```
from numpy import random
a = random.randn(4,4)
dotProduct = a.dot(a)
dotProduct
```

Out[25]:
```
array([[ 1.1495388 ,  0.72712538, -0.50625976, -0.33844933],
       [ 0.40021836, -0.82724181,  0.77992948,  0.4899841 ],
       [-0.52455795,  0.45079691,  0.92544405, -0.03911725],
       [ 1.45778638, -0.28521383,  0.30679524, -1.7963747 ]])
```

In [26]:
```
from numpy.linalg import inv
inverse = inv(a)
inverse
```

Out[26]:
```
array([[ 0.31410666, -0.16374466, -0.56409139, -0.28871586],
       [-0.7177032 , -0.21767584, -0.36724136,  0.6256856 ],
       [-1.00264511, -0.44363705, -0.13009698, -0.1198265 ],
       [ 0.52078608, -0.92722691,  0.1410194 , -0.50359376]])
```

Q15:

15. Store all the created arrays on disk.

In [27]:
```
np.savez('createdarrays.npz', data1=data1, data2=data2, data3=data3, data4=data4,subjects=Subjects,marks=subjectMarks)
```

In [28]:
```
arch = np.load('createdarrays.npz')
print(arch['data1'])
```

```
[ 1  3  9 12 18]
```