GIRIRAJ KRISHNA SHARMA

20051973

[2]:
```
import numpy as np
print(np.__version__)
```

```
1.24.2
```

[3]:
```
# Question : Create a 1D array of numbers from 0 to 9
# Output : #> array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

# Solution
X = np.arange(10)
X
```

[3]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[4]:
```
# Question : Create a 3×3 numpy array of all True's

# Solution
np.full((3,3), True, dtype=bool)

#or
np.full((9), True, dtype=bool).reshape(3,3)

#or
np.ones((3,3), dtype=bool)

#or
np.ones((9), dtype=bool).reshape(3,3)
```

[4]:
```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

[5]:
```
# Question : Extract all odd numbers from array
# input: arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# output: array([1, 3, 5, 7, 9])

#Solution

```

```
arr = np.arange(10)

arr[arr%2 == 1]
```

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

[6]:
```
# Question: Replace all odd numbers in arr with -1
# input: arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# output: array([ 0, -1,  2, -1,  4, -1,  6, -1,  8, -1])

# Solution

arr = np.arange(10)

arr[arr%2 == 1] = -1
arr
```

[6]:
```
array([ 0, -1,  2, -1,  4, -1,  6, -1,  8, -1])
```

[7]:
```
# Question: Replace all odd numbers in arr with -1 without changing arr
# input: arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# output: out
# array([ 0, -1,  2, -1,  4, -1,  6, -1,  8, -1])
# arr
# array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

# Solution

arr = np.arange(10)

out = arr.copy()

out[out%2 == 1] = -1

print('Modified Array')
out

print('\nOriginal Array')
arr
```

```
Modified Array

Original Array
```

[7]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[8]:
```
# Question: Convert a 1D array to a 2D array with 2 rows
# input: np.arange(10)
# output array([[0, 1, 2, 3, 4],
#               [5, 6, 7, 8, 9]])

# Solution

arr = np.arange(10)
arr.reshape(2,5)

# Another solution
arr = np.arange(10)
arr.reshape(2, -1)  # Setting to -1 automatically decides the number of cols
```

[8]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

[9]:
```
# Question: Stack arrays a and b vertically
# input: a = np.arange(10).reshape(2,-1)
#        b = np.repeat(1, 10).reshape(2,-1)

# output: array([[0, 1, 2, 3, 4],
#                [5, 6, 7, 8, 9],
#                [1, 1, 1, 1, 1],
#                [1, 1, 1, 1, 1]])

# Solution

a = np.arange(10).reshape(2,-1)
b = np.repeat(1, 10).reshape(2,-1)

np.vstack([a,b])
```

[9]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
```

[10]:
```
# Question: Stack the arrays a and b horizontally.

# Input: a = np.arange(10).reshape(2,-1)
#        b = np.repeat(1, 10).reshape(2,-1)
# Output: array([[0, 1, 2, 3, 4, 1, 1, 1, 1, 1],
#                [5, 6, 7, 8, 9, 1, 1, 1, 1, 1]])


# Solution:
```

```
a = np.arange(10).reshape(2,-1)
b = np.repeat(1, 10).reshape(2,-1)

np.hstack([a,b])
```

[10]:
```
array([[0, 1, 2, 3, 4, 1, 1, 1, 1, 1],
       [5, 6, 7, 8, 9, 1, 1, 1, 1, 1]])
```

[11]:
```
# Question: Create the following pattern without hardcoding. Use only numpy␣
 ↪functions and the below input array a.

# Input: a = np.array([1,2,3])
# Output: array([1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])


# Solution

a = np.array([1,2,3])
np.r_[np.repeat(a, 3), np.tile(a, 3)]

#other solution
np.hstack((np.repeat(a, 3), np.tile(a, 3)))
```

[11]:
```
array([1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])
```

[12]:
```
# Question: Get the common items between a and b

# Input: a = np.array([1,2,3,2,3,4,3,4,5,6])
#        b = np.array([7,2,10,2,7,4,9,4,9,8])

# Output: array([2, 4])


# Solution:
a = np.array([1,2,3,2,3,4,3,4,5,6])
b = np.array([7,2,10,2,7,4,9,4,9,8])
np.intersect1d(a,b)
```

[12]:
```
array([2, 4])
```

[13]:
```
# Question: From array a remove all items present in array b

# Input: a = np.array([1,2,3,4,5])
#        b = np.array([5,6,7,8,9])

# Output: array([1,2,3,4])
```

```
# Solution
a = np.array([1,2,3,4,5])
b = np.array([5,6,7,8,9])

np.setdiff1d(a,b)
```

[13]:
```
array([1, 2, 3, 4])
```

[14]:
```
# Question: Get the positions where elements of a and b match

# Input: a = np.array([1,2,3,2,3,4,3,4,5,6])
#        b = np.array([7,2,10,2,7,4,9,4,9,8])

# Output: (array([1, 3, 5, 7]),)


# Solution

a = np.array([1,2,3,2,3,4,3,4,5,6])
b = np.array([7,2,10,2,7,4,9,4,9,8])

np.where(a == b)
```

[14]:
```
(array([1, 3, 5, 7], dtype=int64),)
```

[15]:
```
# Question: Get all items between 5 and 10 from a.

# Input: a = np.array([2, 6, 1, 9, 10, 3, 27])
# Output: (array([6, 9, 10]),)


# Solution

a = np.array([2, 6, 1, 9, 10, 3, 27])
a[(a >= 5) & (a <= 10)]
```

[15]:
```
array([ 6,  9, 10])
```

[16]:
```
# Question: Convert the function maxx that works on two scalars, to work on two␣
 ↪arrays.
# Input:

def maxx(x, y):
    if x >= y:
        return x
    else:
```

```
        return y

# maxx(1, 5)
#> 5

# Output:
# a = np.array([5, 7, 9, 8, 6, 4, 5])
# b = np.array([6, 3, 4, 8, 9, 7, 1])
# pair_max(a, b)
# array([ 6.,  7.,  9.,  8.,  9.,  7.,  5.])

# Solution

def pair_max(x, y):
    # here I am using map to make tuple from a and b, other solution is using␣
 ↪zip(a,b)
    maximum = [maxx(a,b) for a,b in map(lambda a,b:(a,b),x,y)]
    # using zip
    # maximum = [maxx(a,b) for a,b in zip(x,y)]
    return np.array(maximum)

a = np.array([5, 7, 9, 8, 6, 4, 5])
b = np.array([6, 3, 4, 8, 9, 7, 1])

pair_max(a,b)
```

[16]:
```
array([6, 7, 9, 8, 9, 7, 5])
```

[17]:
```
# Question: Swap columns 1 and 2 in the array arr.

# Input:

arr = np.arange(9).reshape(3,3)

print('Original array')
arr

# Solution

print("\nModified array")
arr[:, [1,0,2]]
```

```
Original array

Modified array
```

[17]:
```
array([[1, 0, 2],
       [4, 3, 5],
       [7, 6, 8]])
```

[18]:
```
# Question: Swap rows 1 and 2 in the array arr:

# Input:

arr = np.arange(9).reshape(3,3)
print('Original array')
arr

# Solution

print("\nModified array")
arr[[1,0,2], :]
```

```
Original array

Modified array
```

[18]:
```
array([[3, 4, 5],
       [0, 1, 2],
       [6, 7, 8]])
```

[19]:
```
# Question: Reverse the rows of a 2D array arr.

# Input:

arr = np.arange(9).reshape(3,3)

print('Original array')
arr

# Solution

print("\nModified array")
arr[::-1, :]
```

```
Original array

Modified array
```

[19]:
```
array([[6, 7, 8],
       [3, 4, 5],
       [0, 1, 2]])
```

[20]:
```
# Question: Reverse the columns of a 2D array arr.

# Input: arr = np.arange(9).reshape(3,3)

# Solution

arr = np.arange(9).reshape(3,3)
print('Original array')
arr


print("\nModified array")
arr[:, ::-1]
```

```
Original array

Modified array
```

[20]:
```
array([[2, 1, 0],
       [5, 4, 3],
       [8, 7, 6]])
```

[21]:
```
# Question: Create a 2D array of shape 5x3 to contain random decimal numbers␣
 ↪between 5 and 10.

# Solution:

rand_arr = np.random.uniform(5,10, size=(5,3))
rand_arr
```

[21]:
```
array([[6.87002046, 9.61643417, 6.06037858],
       [6.68709556, 9.55509519, 8.84574836],
       [7.77841326, 7.0560554 , 5.36966421],
       [5.13337423, 9.14736216, 9.00253587],
       [5.4988653 , 5.93242979, 8.27820228]])
```

[22]:
```
# Pretty print rand_arr by suppressing the scientific notation (like 1e10)

# Input:
# Create the random array
np.random.seed(100)
rand_arr = np.random.random([3,3])/1e3
np.set_printoptions(suppress=False)
rand_arr

# Output:
#> array([[ 0.000543,  0.000278,  0.000425],
```

```
#>        [ 0.000845,  0.000005,  0.000122],
#>        [ 0.000671,  0.000826,  0.000137]])

np.set_printoptions(suppress=True)
rand_arr
#> array([[ 0.000543,  0.000278,  0.000425],
#>        [ 0.000845,  0.000005,  0.000122],
#>        [ 0.000671,  0.000826,  0.000137]])
```

[22]:
```
array([[0.0005434 , 0.00027837, 0.00042452],
       [0.00084478, 0.00000472, 0.00012157],
       [0.00067075, 0.00082585, 0.00013671]])
```

[23]:
```
# Question: Limit the number of items printed in python numpy array a to a␣
 ↪maximum of 6 elements.
a = np.arange(15)
np.set_printoptions(threshold=6)
a
```

[23]:
```
array([ 0,  1,  2, …, 12, 13, 14])
```

[24]:
```
# Question: Print the full numpy array a without truncating.

# Input: np.set_printoptions(threshold=6)
# a = np.arange(15)
# a

# Output: a
#> array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])

# Solution

a = np.arange(15)


np.set_printoptions(threshold=15)
a
```

[24]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

[25]:
```
# Question: Import the iris dataset keeping the text intact.

# Solution:
iris_data = np.genfromtxt('Iris.csv', delimiter=',', skip_header=1,
                          usecols = [0,1,2,3,4,5], dtype = object)
iris_data
```

[25]:
```
array([[b'1', b'5.1', b'3.5', b'1.4', b'0.2', b'Iris-setosa'],
       [b'2', b'4.9', b'3.0', b'1.4', b'0.2', b'Iris-setosa'],
       [b'3', b'4.7', b'3.2', b'1.3', b'0.2', b'Iris-setosa'],
       …,
       [b'148', b'6.5', b'3.0', b'5.2', b'2.0', b'Iris-virginica'],
       [b'149', b'6.2', b'3.4', b'5.4', b'2.3', b'Iris-virginica'],
       [b'150', b'5.9', b'3.0', b'5.1', b'1.8', b'Iris-virginica']],
      dtype=object)
```

[26]:
```
# Question-26: Extract the text column species from the 1D iris imported in␣
 ↪previous question.

data = np.genfromtxt('Iris.csv', delimiter=',', skip_header=1,
                          usecols = [-1], dtype = object)
data
```

[26]:
```
array([b'Iris-setosa', b'Iris-setosa', b'Iris-setosa', …,
       b'Iris-virginica', b'Iris-virginica', b'Iris-virginica'],
      dtype=object)
```

[27]:
```
# Question: Convert the 1D iris to 2D array iris_2d by omitting the species␣
 ↪text field.
iris_data = np.genfromtxt('Iris.csv', delimiter=',', skip_header=1,␣
 ↪dtype='float', usecols=[0,1,2,3])
iris_data
```

[27]:
```
array([[  1. ,   5.1,   3.5,   1.4],
       [  2. ,   4.9,   3. ,   1.4],
       [  3. ,   4.7,   3.2,   1.3],
       …,
       [148. ,   6.5,   3. ,   5.2],
       [149. ,   6.2,   3.4,   5.4],
       [150. ,   5.9,   3. ,   5.1]])
```

[28]:
```
# Question: Find the mean, median, standard deviation of iris's sepallength␣
 ↪(1st column)

iris_data = np.genfromtxt('Iris.csv', delimiter=',', skip_header=1, usecols =␣
 ↪[1])

print('Mean', np.mean(iris_data))
print('Median', np.median(iris_data))
print('Standard Deviation', np.std(iris_data))
```

```
Mean 5.843333333333334
Median 5.8
Standard Deviation 0.8253012917851409
```

[29]:
```
# Question: Create a normalized form of iris's sepallength whose values range
#           exactly between 0 and 1 so that the minimum has value 0 and maximum␣
 ↪has value 1.

# Solution

iris_data = np.genfromtxt('Iris.csv', delimiter=',', dtype='float',␣
 ↪usecols=[1], skip_header=1)

(iris_data - np.min(iris_data))/(np.max(iris_data) - np.min(iris_data))
```

[29]:
```
array([0.22222222, 0.16666667, 0.11111111, …, 0.61111111, 0.52777778,
       0.44444444])
```

[30]:
```
# Question: Compute the softmax score of sepallength.

iris_data = np.genfromtxt('Iris.csv', delimiter=',', dtype='float',␣
 ↪usecols=[1], skip_header=1)
softmax = np.exp(iris_data)/sum(np.exp(iris_data))
softmax.sum() # it must sum 1
softmax
```

[30]:
```
array([0.00221959, 0.00181724, 0.00148783, …, 0.00900086, 0.006668  ,
       0.00493978])
```

[31]:
```
# Question. Find the 5th and 95th percentile of iris's sepallength
iris_data = np.genfromtxt('Iris.csv', delimiter=',', dtype='float',␣
 ↪usecols=[1], skip_header=1)

np.percentile(iris_data, q=[5, 95])
```

[31]:
```
array([4.6  , 7.255])
```

[32]:
```
# Question: Insert np.nan values at 20 random positions in iris_2d dataset

# Solution
iris_data = np.genfromtxt('Iris.csv', delimiter=',', dtype='float',␣
 ↪usecols=[1,2,3,4], skip_header=1)
for i in np.random.randint(0, len(iris_data), 20):
    iris_data[i]=np.nan
iris_data
```

[32]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       …,
       [6.5, 3. , 5.2, 2. ],
```

```
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

[33]:
```
# Question: Find the number and position of missing values in iris_2d's␣
 ↪sepallength (1st column)

iris_data = np.genfromtxt('Iris.csv', delimiter=',', dtype='float',␣
 ↪usecols=[1,2,3,4], skip_header=1)
iris_data[np.random.randint(len(iris_data), size=20),np.random.
 ↪randint(4,size=20)] = np.nan

# Find total mising value in complete data
print("Number of missing values in Iris data: \n", np.isnan(iris_data[:, :]).
 ↪sum())

# Find total mising value in 1D data
print("Number of missing values in any one feature of Iris data: \n", np.
 ↪isnan(iris_data[:, 0]).sum())

print("Position of missing values: \n", np.where(np.isnan(iris_data[:, 0])))
```

```
Number of missing values in Iris data:
 20
Number of missing values in any one feature of Iris data:
 5
Position of missing values:
 (array([ 38,  80, 106, 113, 121], dtype=int64),)
```

[34]:
```
# Question: Filter the rows of iris_2d that has petallength (3rd column) > 1.5␣
 ↪and sepallength (1st column) < 5.0

iris_data = np.genfromtxt('Iris.csv', delimiter=',', dtype='float',␣
 ↪usecols=[1,2,3,4], skip_header=1)

# Solution
iris_data[(iris_data[:, 2] > 1.5) & (iris_data[:, 0] < 5.0)]
```

[34]:
```
array([[4.8, 3.4, 1.6, 0.2],
       [4.8, 3.4, 1.9, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [4.9, 2.4, 3.3, 1. ],
       [4.9, 2.5, 4.5, 1.7]])
```

[35]:
```
# Question: Select the rows of iris_2d that does not have any nan value.

diabetes_data = np.genfromtxt('diabetes.csv', delimiter=',',
```

```
                                dtype='float', usecols=[0,1,2,3,4,5,6,7],␣
 ↪skip_header=1)
diabetes_data[np.random.randint(150, size=20), np.random.randint(4, size=20)] =␣
 ↪np.nan
diabetes_data[np.sum(np.isnan(diabetes_data), axis = 1) == 0][:5]
```

[35]:
```
array([[  1.   ,  85.   ,  66.   , …,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , …,  23.3  ,   0.672,  32.   ],
       [  1.   ,  89.   ,  66.   , …,  28.1  ,   0.167,  21.   ],
       [  0.   , 137.   ,  40.   , …,  43.1  ,   2.288,  33.   ],
       [  3.   ,  78.   ,  50.   , …,  31.   ,   0.248,  26.   ]])
```

[36]:
```
# question: Find the correlation between SepalLength(1st column) and␣
 ↪PetalLength(3rd column) in iris_2d
# insted or using iris data I am going to used pima diabetes data and going to␣
 ↪find corelation
# between BP(1st column) and BMI (5th column).

diabetes_data = np.genfromtxt('diabetes.csv',
                              delimiter=',', dtype='float',␣
 ↪usecols=[0,1,2,3,4,5,6,7], skip_header=1)

print(np.corrcoef(diabetes_data[:, 1], diabetes_data[:, 5]))

print('\n')
# you can get correlation by getting value at index [0,1] or [1,0]
print(np.corrcoef(diabetes_data[:, 1], diabetes_data[:, 5])[0,1])
```

```
[[1.         0.22107107]
 [0.22107107 1.        ]]


0.2210710694589828
```

[37]:
```
# Question: What is the value of second longest petallength of species setosa
# For this question I am going to find second highest bloodpressure (2nd␣
 ↪column) where outcome is 1
diabetes_data = np.genfromtxt('diabetes.csv',
                              delimiter=',', dtype=object,␣
 ↪usecols=[0,1,2,3,4,5,6,7,8], skip_header=1)

# Solution
bloodpressure= diabetes_data[diabetes_data[:, 8]==b'1', [2]].astype('float')

np.unique(np.sort(bloodpressure))[-2]
```

[37]:
```
110.0
```

[38]:
```
# Question: Sort the iris dataset based on sepallength column.
# In this problem, I am going to sort the diabetes dataset based on Glucose␣
 ↪(1th column)
diabetes_data = np.genfromtxt('diabetes.csv',
                              delimiter=',', dtype=object,␣
 ↪usecols=[0,1,2,3,4,5,6,7,8], skip_header=1)

diabetes_data[diabetes_data[:,1].argsort()]
```

[38]:
```
array([[b'1', b'0', b'74', …, b'0.299', b'21', b'0'],
       [b'5', b'0', b'80', …, b'0.346', b'37', b'1'],
       [b'1', b'0', b'68', …, b'0.389', b'22', b'0'],
       …,
       [b'4', b'99', b'72', …, b'0.294', b'28', b'0'],
       [b'4', b'99', b'76', …, b'0.223', b'21', b'0'],
       [b'2', b'99', b'60', …, b'0.453', b'21', b'0']], dtype=object)
```

[39]:
```
# Question: Find the most frequent value of petal length (3rd column) in iris␣
 ↪dataset.

iris_data = np.genfromtxt('Iris.csv', delimiter=',',
                          dtype=object, usecols=[1,2,3,4,5], skip_header=1)

v,c = np.unique(iris_data[:, 2], return_counts=True)
v[np.argmax(c)]
```

[39]:
```
b'1.5'
```

[40]:
```
# Question: Find the position of the first occurrence of a value greater than 1.
 ↪0 in
# petalwidth 4th column of iris dataset.
iris_data = np.genfromtxt('Iris.csv', delimiter=',',
                          dtype=object, usecols=[4], skip_header=1)

np.argwhere(iris_data[:].astype(float) > 1.0)[0]
```

[40]:
```
array([50], dtype=int64)
```

[41]:
```
# Question: From the array a, replace all values greater than 30 to 30 and less␣
 ↪than 10 to 10.

# Solution

np.set_printoptions(precision=2)
np.random.seed(100)
a = np.random.uniform(1,50, 20)
```

```
a[a<10]=10
a[a>30]=30
np.set_printoptions(threshold=20)
a
```

[41]:
```
array([27.63, 14.64, 21.8 , 30.  , 10.  , 10.  , 30.  , 30.  , 10.  ,
       29.18, 30.  , 11.25, 10.08, 10.  , 11.77, 30.  , 30.  , 10.  ,
       30.  , 14.43])
```

[42]:
```
# Question: Get the positions of top 5 maximum values in a given array a.
np.random.seed(100)
a = np.random.uniform(1,50, 20)
a
sort = a.argsort()
print('Positions')
sort[-5:][::-1]
print('Values')
a[sort][-5:][::-1]
```

```
Positions
Values
```

[42]:
```
array([48.95, 44.67, 42.39, 41.47, 41.  ])
```

[43]:
```
# Question: Compute the counts of unique values row-wise.

# Solution
def counts_of_all_values_rowwise(arr2d):
    # Unique values and its counts row wise
    num_counts_array = [np.unique(row, return_counts=True) for row in arr2d]

    # Counts of all values row wise
    return([[int(b[a==i]) if i in a else 0 for i in np.unique(arr2d)] for a, b␣
 ↪in num_counts_array])

np.random.seed(100)
np.set_printoptions(threshold=10)
arr = np.random.randint(1,11,size=(6, 10))
arr
print(np.arange(1,11))
counts_of_all_values_rowwise(arr)
```

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

[43]:
```
[[1, 0, 2, 1, 1, 1, 0, 2, 2, 0],
 [2, 1, 3, 0, 1, 0, 1, 0, 1, 1],
 [0, 3, 0, 2, 3, 1, 0, 1, 0, 0],
```

```
 [1, 0, 2, 1, 0, 1, 0, 2, 1, 2],
 [2, 2, 2, 0, 0, 1, 1, 1, 1, 0],
 [1, 1, 1, 1, 1, 2, 0, 0, 2, 1]]
```

[44]:
```
# Question: Convert array_of_arrays into a flat linear 1d array.
arr1 = np.arange(3)
arr2 = np.arange(3,7)
arr3 = np.arange(7,10)

arr_2d = np.concatenate([arr1, arr2, arr3])
print(arr_2d)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

[45]:
```
# Q-51

#1 How to get the n largest values of an array ?
Z = np.arange(10000)
np.random.shuffle(Z)
n = 5

# Slow
print (Z[np.argsort(Z)[-n:]])

# Fast
print (Z[np.argpartition(-Z,n)[:n]])
```

```
[9995 9996 9997 9998 9999]
[9999 9998 9996 9997 9995]
```

[46]:
```
#2 Given an arbitrary number of vectors, build the cartesian product
#  (every combinations of every item).
def cartesian(arrays):
    arrays = [np.asarray(a) for a in arrays]
    shape = (len(x) for x in arrays)

    ix = np.indices(shape, dtype=int)
    ix = ix.reshape(len(arrays), -1).T

    for n, arr in enumerate(arrays):
        ix[:, n] = arrays[n][ix[:, n]]

    return ix

print (cartesian(([1, 2, 3], [4, 5], [6, 7])))
```

```
[[1 4 6]
 [1 4 7]
```

```
 [1 5 6]
 …
 [3 4 7]
 [3 5 6]
 [3 5 7]]
```

[47]:
```
#3 Consider a 16x16 array, how to get the blocksum (block size is 4x4)?
Z = np.ones((16,16))
k = 4
S = np.add.reduceat(np.add.reduceat(Z, np.arange(0, Z.shape[0], k), axis=0), np.
 ↪arange(0, Z.shape[1], k), axis=1)
print ('input array')
print (Z)
print ('block sum')
print (S)
```

```
input 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. … 1. 1. 1.]
 [1. 1. 1. … 1. 1. 1.]]
block sum
[[16. 16. 16. 16.]
 [16. 16. 16. 16.]
 [16. 16. 16. 16.]
 [16. 16. 16. 16.]]
```

[48]:
```
#4 Compute a matrix rank.
Z = np.random.uniform(0,1,(10,10))
U, S, V = np.linalg.svd(Z)    # Singular Value Decomposition
rank = np.sum(S > 1e-10)
print (rank)
```

```
10
```

[49]:
```
#5 How to find the most frequent value in an array?
Z = np.random.randint(0,10,50)
print (Z)
print('rank:', np.bincount(Z).argmax())
```

```
[3 8 7 … 9 4 5]
rank: 3
```

[50]:
```
#6 Extract all the contiguous 3x3 blocks from a random 10x10 matrix.
Z = np.random.randint(0,5,(6,6))
```

```
n = 3
i = 1 + (Z.shape[0]-3)
j = 1 + (Z.shape[1]-3)
C = np.lib.stride_tricks.as_strided(Z, shape=(i, j, n, n), strides=Z.strides +␣
 ↪Z.strides)
print(C)
```

```
[[[[2 3 1]
   [0 3 3]
   [4 0 2]]

  [[3 1 2]
   [3 3 0]
   [0 2 2]]

  [[1 2 1]
   [3 0 0]
   [2 2 1]]

  [[2 1 2]
   [0 0 1]
   [2 1 3]]]


 [[[0 3 3]
   [4 0 2]
   [0 1 0]]

  [[3 3 0]
   [0 2 2]
   [1 0 1]]

  [[3 0 0]
   [2 2 1]
   [0 1 1]]

  [[0 0 1]
   [2 1 3]
   [1 1 4]]]


 [[[4 0 2]
   [0 1 0]
   [1 2 0]]

  [[0 2 2]
   [1 0 1]
```

```
  [2 0 4]]

 [[2 2 1]
  [0 1 1]
  [0 4 4]]

 [[2 1 3]
  [1 1 4]
  [4 4 2]]]


[[[0 1 0]
  [1 2 0]
  [0 1 1]]

 [[1 0 1]
  [2 0 4]
  [1 1 4]]

 [[0 1 1]
  [0 4 4]
  [1 4 0]]

 [[1 1 4]
  [4 4 2]
  [4 0 2]]]]
```