NumPy Exercises

```
import numpy as np
```

```
#Create an array of 10 zeros
np.zeros(10)
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
#Create an array of 10 ones
np.ones(10)
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
#Create an array of 10 fives
np.zeros(10)+5
```

```
array([5., 5., 5., 5., 5., 5., 5., 5., 5., 5.])
```

```
#Create an array of integers from 10 to 50
np.arange(10,51)
```

```
array([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, 50])
```

```
#Create an array of all the even integers from 10 to 50
np.arange(10,51,2)
```

```
array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
       44, 46, 48, 50])
```

```
#Create a 3x3 matrix with values ranging from 0 to 8
np.arange(0,9).reshape(3,3)
```

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

```
#Create a 3x3 identity matrix
np.identity(3)
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
#Use NumPy to generate a random number between 0 and 1
np.random.random_sample()
```

```
0.9045917555531892
```

```
#Use NumPy to generate an array of 25 random numbers sampled from a standard normal distri
```

```
np.random.normal(0,1,25)
```

```
array([ 0.0761244 ,  0.1657106 , -2.07544448,  1.11100132, -0.44322636,
       -0.75774521,  0.78573575, -0.10795619, -0.47610658,  1.02725653,
        1.34833681,  0.14039769,  1.76068685, -0.41477958,  0.09575036,
       -0.18744107,  2.30778475,  0.91000422,  0.59159808, -0.21061308,
        1.29201925,  1.1033813 , -1.09567205, -1.49293357,  0.26549478])
```

```
"""
Create the following matrix:
array([[ 0.01,  0.02,  0.03,  0.04,  0.05,  0.06,  0.07,  0.08,  0.09,  0.1 ],
       [ 0.11,  0.12,  0.13,  0.14,  0.15,  0.16,  0.17,  0.18,  0.19,  0.2 ],
       [ 0.21,  0.22,  0.23,  0.24,  0.25,  0.26,  0.27,  0.28,  0.29,  0.3 ],
       [ 0.31,  0.32,  0.33,  0.34,  0.35,  0.36,  0.37,  0.38,  0.39,  0.4 ],
       [ 0.41,  0.42,  0.43,  0.44,  0.45,  0.46,  0.47,  0.48,  0.49,  0.5 ],
       [ 0.51,  0.52,  0.53,  0.54,  0.55,  0.56,  0.57,  0.58,  0.59,  0.6 ],
       [ 0.61,  0.62,  0.63,  0.64,  0.65,  0.66,  0.67,  0.68,  0.69,  0.7 ],
       [ 0.71,  0.72,  0.73,  0.74,  0.75,  0.76,  0.77,  0.78,  0.79,  0.8 ],
       [ 0.81,  0.82,  0.83,  0.84,  0.85,  0.86,  0.87,  0.88,  0.89,  0.9 ],
       [ 0.91,  0.92,  0.93,  0.94,  0.95,  0.96,  0.97,  0.98,  0.99,  1.  ]])
"""
np.arange(0.01, 1.01, 0.01).reshape(10,10)
```

```
array([[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 ],
       [0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2 ],
       [0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3 ],
       [0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4 ],
       [0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48, 0.49, 0.5 ],
       [0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58, 0.59, 0.6 ],
       [0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69, 0.7 ],
       [0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8 ],
       [0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9 ],
       [0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.  ]])
```

```
"""Create an array of 20 linearly spaced points between 0 and 1:
array([ 0.        ,  0.05263158,  0.10526316,  0.15789474,  0.21052632,
        0.26315789,  0.31578947,  0.36842105,  0.42105263,  0.47368421,
        0.52631579,  0.57894737,  0.63157895,  0.68421053,  0.73684211,
        0.78947368,  0.84210526,  0.89473684,  0.94736842,  1.        ])"""
np.linspace(0,1,20)
```

```
array([0.        , 0.05263158, 0.10526316, 0.15789474, 0.21052632,
       0.26315789, 0.31578947, 0.36842105, 0.42105263, 0.47368421,
       0.52631579, 0.57894737, 0.63157895, 0.68421053, 0.73684211,
       0.78947368, 0.84210526, 0.89473684, 0.94736842, 1.        ])
```

Numpy Indexing and Selection

```
mat = np.arange(1,26).reshape(5,5)
mat
```

```
array([[ 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]])
```

```
# array([[12, 13, 14, 15], [17, 18, 19, 20], [22, 23, 24, 25]])
mat[2:,1:]
```

```
array([[12, 13, 14, 15],
       [17, 18, 19, 20],
       [22, 23, 24, 25]])
```

```
#20
mat[3,4]
```

```
20
```

```
#array([[ 2], [ 7], [12]])
mat[:3,1]
```

```
array([ 2,  7, 12])
```

```
#array([21, 22, 23, 24, 25])
mat[4]
```

```
array([21, 22, 23, 24, 25])
```

```
#array([[16, 17, 18, 19, 20], [21, 22, 23, 24, 25]])
mat[3:]
```

```
array([[16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
```

```
'''Get the sum of all the values in mat
325'''
np.sum(mat)
```

```
325
```

```
'''Get the standard deviation of the values in mat
7.2111025509279782'''
np.std(mat)
```

```
7.211102550927978
```

```
'''Get the sum of all the columns in mat
array([55, 60, 65, 70, 75])'''
mat.sum(axis=0)
```

```
array([55, 60, 65, 70, 75])
```

```
'''Find median values in all columns
[11. 12. 13. 14. 15.]'''
np.median(mat, axis=0)
```

```
array([11., 12., 13., 14., 15.])
```

```
'''Find average values in all columns
‘[11. 12. 13. 14. 15.]’'''
np.average(mat, axis=0)
```

```
array([11., 12., 13., 14., 15.])
```

```
'''Find median values in all rows
[ 3. 8. 13. 18. 23.]'''
np.median(mat, axis=1)
```

```
array([ 3.,  8., 13., 18., 23.])
```

```
'''Find average values in all rows
[ 3. 8. 13. 18. 23.]'''
np.average(mat, axis=1)
```

```
array([ 3.,  8., 13., 18., 23.])
```

Matplotlib exercise

```
from google.colab import files
uploaded = files.upload()
```

Wybierz pliki Crime_Data_..._small.csv

- **Crime_Data_from_2010_small.csv**(application/vnd.ms-excel) - 402860 bytes, last modified: 28.12.2018 - 100% done

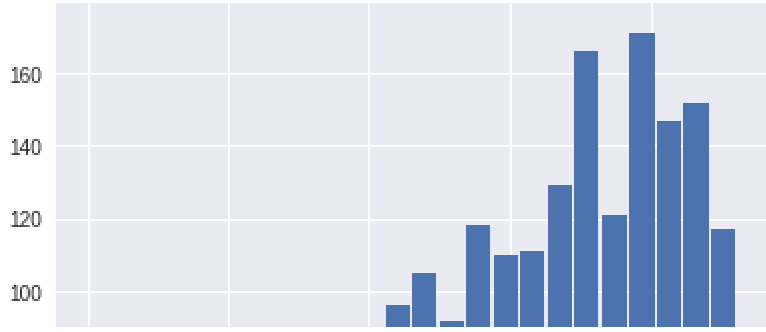
```
Saving Crime_Data_from_2010_small.csv to Crime_Data_from_2010_small.csv
```

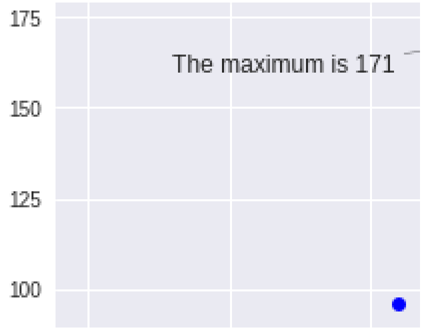
```
import csv
import collections
with open('Crime_Data_from_2010_small.csv') as f:
  data=[]
  hours=[]
  reader = csv.reader(f, delimiter=',')
  row0 = next(reader)
  for row in reader:
    data.append(row)
    hours.append(int(row[3])//100)

c = collections.Counter(hours)
c = sorted(c.items())
print(c)
```

```
[(0, 45), (1, 52), (2, 50), (3, 19), (4, 19), (5, 7), (6, 12), (7, 20), (8, 30), (9
```

```
import matplotlib.pyplot as plt

fig, ax= plt.subplots(nrows=1, ncols=2, figsize=[14,6])
ax[0].hist(hours, rwidth=0.9, bins=24)
X,Y=zip(*c)
ax[1].plot(X,Y, 'bo')
ax[1].annotate(f'The maximum is {max(Y)}', xy=(19,171), xytext=(3,160), arrowprops=dict(ar
```

```
Text(3,160,'The maximum is 171')
```





Implement a Gaussian elimination algorithm with a function taking two arguments: A - the matrix, b - right hand side vector

```python
import numpy as np
def gaussian(A,b):
  myM=np.column_stack((A,b))
  myM=myM.astype(float)
  for q in range(myM.shape[0]):
    for w in range(q+1,myM.shape[0]):
      myM[w,q:]=myM[q,q:]*(myM[w,q]/myM[q,q]*(-1))+myM[w,q:]
  return myM

A = np.arange(1, 17, dtype=np.float64).reshape(4,4)
A[1,2] = 88
A[1,3] = -3
A[2,3] = -3
print(f'A = {A}')

x = np.ones(A.shape[0])
print(f'Original x = {x}')
b = A  @ x.T
print(f'Right hand side for testing: b = {b}')


Ae = gaussian(A, b)
print(f'Check if A was unchanged ')
print(f'Eliminated augmented matrix:\n {Ae}')
print(f'Eliminated augmented matrix A part: {Ae[:,:-1]}')
print(f'Eliminated augmented matrix b part: {Ae[:,Ae.shape[1]-1]}')

# Find solution
#x = back(Ae[:,:-1],Ae[:,Ae.shape[1]-1])
#print(f'Solution: {x}')
#there is a problem with 'back' function - "name 'back' is not defined"
```

```
A = [[ 1.  2.  3.  4.]
 [ 5.  6. 88. -3.]
 [ 9. 10. 11. -3.]
 [13. 14. 15. 16.]]
Original x = [1. 1. 1. 1.]
Right hand side for testing: b = [10. 96. 27. 58.]
Check if A was unchanged
Eliminated augmented matrix:
 [[   1.     2.     3.     4.    10. ]
 [   0.    -4.    73.   -23.    46. ]
 [   0.     0.  -162.     7.  -155. ]
 [   0.     0.     0.    22.5   22.5]]
Eliminated augmented matrix A part: [[   1.     2.     3.     4. ]
 [   0.    -4.    73.   -23. ]
 [   0.     0.  -162.     7. ]
 [   0.     0.     0.    22.5]]
Eliminated augmented matrix b part: [  10.    46.  -155.    22.5]
```