AI for Biotechnology
Exercise 1

Prof. Dr. Dominik Grimm
Bioinformatics Research Lab
TUM Campus Straubing for Biotechnology and Sustainability

Exercise E1.1

Have a look at the following Python code:

```
import numpy as np
x = np.array([1,2,3], [4,5,6])
```

Why does this code snippet not create a 2-dimensional array?

Solution: The code does not create a 2-dimensional array because the sequence of numbers must be passed as a single argument. In this example we pass two arguments - two lists - to the function `np.array`. However, we must pass a nested list: `x = np.array([[1,2,3],[4,5,6]])`

Exercise E1.2

Have a look at the following Python code:

```
import numpy as np
x = np.array([1,2,3])
y = np.array([[1,2,3]])
```

What is the difference between the `x` and `y`?

Solution: `x` is a one-dimensional array with three elements, whereas `y` is a 1×3 array (2-dimensional).

Exercise E1.3

Execute the following code:

```
import numpy as np
X = np.linspace(1,96,96).reshape(3,8,4)
```

Answer the following questions using Python:

a) How many dimensions has the array `X`, how many elements does the array have and what is the size of each dimension?

```
import numpy as np
X = np.linspace(1,96,96).reshape(3,8,4)
print("Dimensions: " + str(X.ndim)) #3
print("Size: " + str(X.size)) #96
print("Shape: " + str(X.shape)) #3, 8, 4
```

b) Use indexing or slicing to obtain the element 44.0

```
X[1,2,-1]
```

c) Use indexing or slicing to obtain `[13., 14., 15., 16.]`

```
X[0,3,:]
```

d) Use indexing or slicing to obtain `[2., 6., 10., 14., 18., 22., 26., 30.]`

```
X[0,:,1]
```

e) Use indexing or slicing to obtain the 3×3 array

```
[[74., 75., 76.],
 [78., 79., 80.],
 [82., 83., 84.]]
```

```
X[2,2:5,1:4]
```

f) Use indexing or slicing to obtain the 3×4 array

```
[[ 9., 10., 11., 12.],
 [17., 18., 19., 20.],
 [25., 26., 27., 28.]]
```

```
X[0,[2,4,6],:]
```

g) Use indexing or slicing to obtain the 2×2 array

```
[[ 9., 10.],
 [41., 42.]]
```

```
X[[0,1],2,:2]
```

Exercise E1.4

Given is the following Python code to draw 100 random samples from a Gaussian distribution with zero mean ($\mu = 0$) and a standard deviation of $\sigma = 10$:

```
np.random.seed(41)
x = np.random.normal(0,10,100)
```

Use boolean indexing to return a sub-array that only contains values that are less or equal than 1. What is the size of the new sub-array?

```
np.random.seed(41)
x = np.random.normal(0,10,100)
sub_array = x[x<=1]
print("Size sub-array: " + str(sub_array.size))
```

Exercise E1.5

The Gaussian function with mean μ and standard deviation σ is defined as follows:

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1}$$

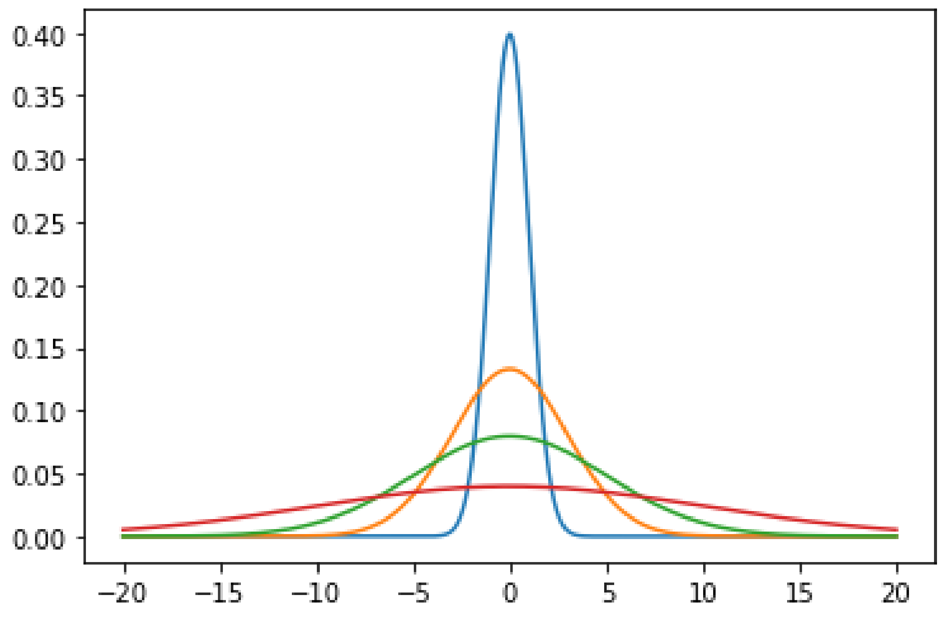
Write a Python function with the name **gaussian** to calculate the Gaussian function for a given mean, standard deviation and a list of x-values. Further, compute and plot the distributions of the Gaussians for $\mu = 0$ and four different standard deviations $\sigma \in \{1, 3, 5, 10\}$. Use a grid of 10'000 x-values in the interval of $-20 \leq x \leq 20$.

```python
import numpy
%matplotlib inline
import pylab as pl
#gaussian function
def gaussian(x, std, mean):
    return (1.0/(std*np.sqrt(2.0*np.pi)) *
            np.exp(-((x-mean)**2)/(2*std**2)))

mean = 0
stds = np.array([1,3,5,10])
x = np.linspace(-20, 20, 10000)

#with for loop
for std in stds:
    y = gaussian(x,std,mean)
    pl.plot(x,y)

#alternative without for loop
x = x[:,np.newaxis]
y = gaussian(x,stds,mean)
pl.plot(x,y)
```



Exercise E1.6

In this exercise we will use the Python library `scipy`. This library contains additional functionality important for scientific computing, e.g. advanced solvers for optimisation problems. For this exercise we will just use the library to load a RGB image. Images are stored in a simple 3-dimensional array $m \times n \times 3$. Each of the three $m \times n$ arrays represents one of the three colour planes, as illustrated in Figure 1.

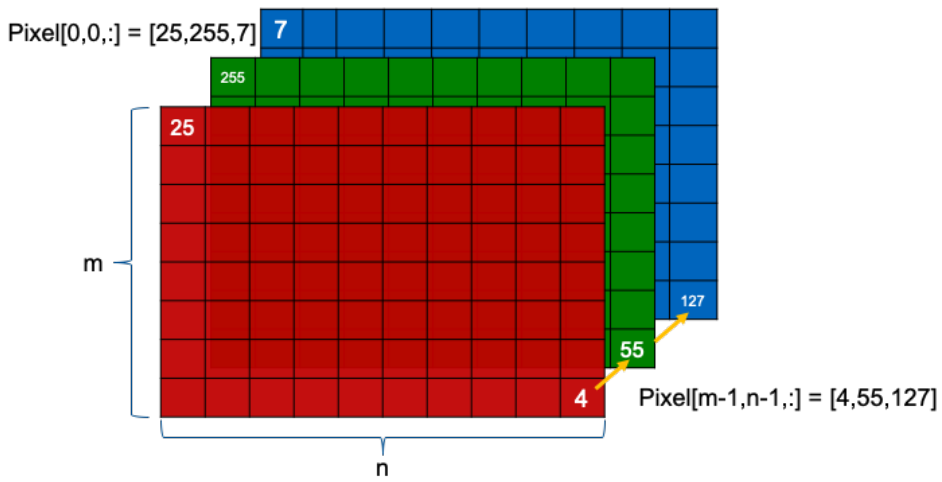


Figure 1: Representation of an RGB image as NumPy array

Values within the array range between 0 and 255, representing the color intensity of the pixel at position (i,j). The following code will load and plot the image, as illustrated in Figure 2.

```
import scipy.misc as misc
%matplotlib inline
import pylab as pl
im = misc.face()
pl.imshow(im)
```

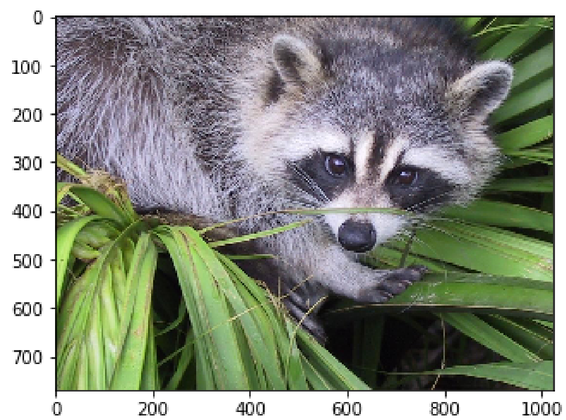


Figure 2: RGB image plotted using matplotlib

Answer the following questions and solve the tasks:

a) What is the shape of the image im?

```
print("Shape of image: ", im.shape)
```

Output: Shape of image: (768, 1024, 3)
Thus the image has a size of 768 pixel times 1024 pixel and three layers, one for each color

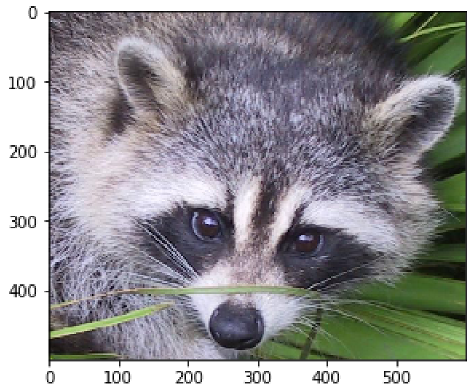
b) How many pixels has the image?

```
number_pixels = im.shape[0]*im.shape[1]
print("The Image has a total of %d pixels"%number_pixels)
```

Output: The Image has a total of 786432 pixel

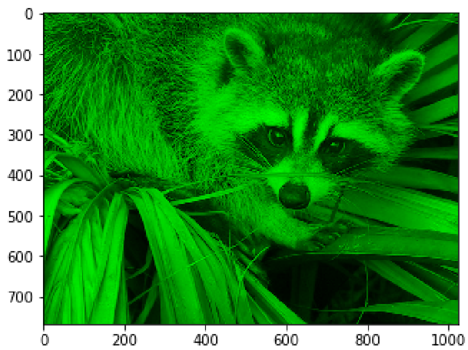
c) Use indexing and slicing to crop the image, such that you get a closer look at the face of the animal. Use the plotting functionality to check your results!

```
#Coordinates are just an example, yours might be different.
pl.imshow(im[0:500,350:950,:])
```



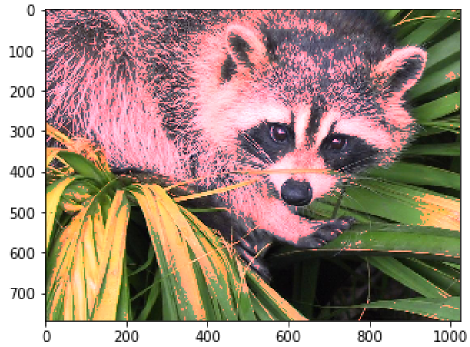
d) Create a copy of your image using the command new_im = im.copy(). Set all intensity values in the copy to zero, except for the green plane and visualise your image.

```
new_im = im.copy()
new_im[:,:,[0,2]]=0
pl.imshow(new_im)
```



e) Again, create a copy of your original image using the command `new_im = im.copy()`. Use boolean indexing to set all values for the red plane to 255 that are greater than 127.

```
new_im = im.copy()
new_im[new_im[:,:,0]>127,0] = 255
pl.imshow(new_im)
```



Exercise E1.7

Given are the two matrices $\boldsymbol{A} \in \mathbb{R}^{3\times 2}$ and $\boldsymbol{B} \in \mathbb{R}^{3\times 2}$:

$$\boldsymbol{A} = \begin{pmatrix} -2 & 3 \\ 4 & 1 \\ -1 & 5 \end{pmatrix} \text{ und } \boldsymbol{B} = \begin{pmatrix} 1 & 4 \\ 0 & -2 \\ 3 & 5 \end{pmatrix}$$

and the two vectors $\boldsymbol{x} = (8, -5)^\top$ und $\boldsymbol{y} = (3, 2)^\top$. Compute the following expressions using the `NumPy` library:

a) $\boldsymbol{A} + \boldsymbol{B}$

b) $\boldsymbol{A}(\boldsymbol{x} + \boldsymbol{y})$

c) $(\boldsymbol{A} + \boldsymbol{B})\boldsymbol{x}$

d) $\boldsymbol{x}^\top \boldsymbol{y}$

e) $5\boldsymbol{A}\boldsymbol{B}^\top$

```python
A = np.array([[-2,3],
              [ 4,1],
              [-1,5]])
B = np.array([[1, 4],
              [0, -2],
              [3, 5]])
x = np.array([[8,-5]]).T
y = np.array([[3,2]]).T
print("a)")
print(A+B)
print("b)")
print(A.dot(x+y))
print("c)")
print((A+B).dot(x))
print("d)")
print((x.T).dot(y))
print("e)")
print(5*A.dot(B.T))
```

Output:

```
a)
[[-1  7]
 [ 4 -1]
 [ 2 10]]
b)
[[-31]
 [ 41]
 [-26]]
c)
[[-43]
 [ 37]
 [-34]]
d)
[[14]]
e)
[[ 50 -30  45]
 [ 40 -10  85]
 [ 95 -50 110]]
```

Exercise E1.8

Find out yourself how to compute the inverse of a matrix using the `NumPy` library and invert the following matrices:

a) $\boldsymbol{A} = \begin{pmatrix} 1 & 4 & -1 \\ -1 & -3 & 5 \\ 5 & 19 & -8 \end{pmatrix}$

b) $\boldsymbol{B} = \begin{pmatrix} 0 & 1 & -2 \\ 1 & 1 & 0 \\ 2 & 1 & 1 \end{pmatrix}$

```
A = np.array([[1,4,-1],
              [-1,-3,5],
              [5,19,-8]])
B = np.array([[0,1,-2],
              [1,1,0],
              [2,1,1]])
print("Inverse of A:")
print(np.linalg.inv(A))
print("Inverse of B:")
print(np.linalg.inv(B))
```

Output:

```
Inverse of A:
[[-71.  13.  17.]
 [ 17.  -3.  -4.]
 [ -4.   1.   1.]]
Inverse of B:
[[ 1. -3.  2.]
 [-1.  4. -2.]
 [-1.  2. -1.]]
```

Exercise E1.9

Given are the two matrices $\boldsymbol{X} \in \mathbb{C}^{3\times3}$ and $\boldsymbol{Y} \in \mathbb{C}^{3\times3}$:

$$\boldsymbol{X} = \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{i}{\sqrt{2}} & -\frac{i}{\sqrt{2}} \\ 1 & 0 & 0 \end{pmatrix} \text{ und } \boldsymbol{Y} = \begin{pmatrix} 2 & i & 0 \\ -i & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

Compute the following expression using the `NumPy` library (Hint: complex numbers have the type `np.complex` and complex numbers are written as `1.j` in `NumPy`):

$$\boldsymbol{R} = \overline{\boldsymbol{X}}^{\top} \boldsymbol{Y} \boldsymbol{X},$$

where $\overline{\boldsymbol{X}}$ is the complex conjugate of $\boldsymbol{X}$. Print the matrix $\boldsymbol{R}$, the real values of $\boldsymbol{R}$ and the diagonal of the real values of $\boldsymbol{R}$.

```
X = np.array([[0,1/np.sqrt(2),1/np.sqrt(2)],
              [0,1.j/np.sqrt(2),-1.j/np.sqrt(2)],
              [1,0,0]],dtype=np.complex)
Y = np.array([[2,1.j,0],
              [-1.j,2,0],
              [0,0,2]],dtype=np.complex)
R = ((X.conj()).T).dot(Y).dot(X)
print("Result Matrix")
print(R)
print("Real Result Matrix")
print(R.real)
print("Diagonal of Real Result Matrix")
print(R.diagonal().real)
```

Output:

```
Result Matrix
[[2.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 1.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 3.+0.j]]
Real Result Matrix
[[2. 0. 0.]
 [0. 1. 0.]
 [0. 0. 3.]]
Diagonal of Real Result Matrix
[2. 1. 3.]
```