1 NumPy

For more see NumPy API
Prerequisites: `import numpy as np`

1.1 Arrays

1.1.1 Input

`x = np.array([0,1,2])`	Create a rank 1 array
`M = np.array([[0,1], [1,0]])`	Create a rank 2 array
`x = np.arange(0,3,0.5)`	Creates array `x = [0, 0.5, 1, 1.5, 2, 2.5]`

1.1.2 Special Matrices

`Z = np.zeros((m,n))`	Definition of a $m \times n$ array of zeros
`O = np.ones((m,n))`	Definition of a $m \times n$ array of ones
`E = np.eye(n)`	Definition of a $n \times n$ identity matrix
`R = np.random.rand(m,n)`	Definition of a $m \times n$ array with uniformly distributed random numbers between 0 and 1
`R = np.random.randn(m,n)`	Definition of a $m \times n$ array with normal distributed random numbers with mean 0 and standard deviation 1

1.1.3 Acces to Matrix Elements

`M[i,j]`	Element at row i and column j
`M[i,:]`	All elements of row i
`M[1:3,i:j]`	Array of rows 1 to 2 and colums i to j
`M[L]`	All elements of M, which have the logical value `True` in L
`M[x]`	Returns array of elements of M according to the values in x.

1.1.4 Operations

`A+B`	Addition
`A-B`	Subtraction
`A*B`	Element-wise multiplication: $a_{ij} \cdot b_{ij}$
`np.matmul(A,B)`	Matrix product of two arrays
`A/B`	Element-wise division: a_{ij}/b_{ij}
`A//B`	Element-wise integer division
`x = np.linalg.solve(A,b)`	Solves linear equation $Ax = b$ for x
`np.transpose(A)`	Transpose array
`A**x`	Element-wise power: a_{ij}^x

1.1.5 Matrix Dimensions and Data Types

`dim = A.ndim`	Get number of dimensions of array
`(m,n) = A.shape`	Get dimensions of rank 2 array A
`s = A.shape`	Dimension size of the array A with any number of dimensions
`A.dtype='uint8'`	Cast array A to uint8

1.1.6 Miscellaneous

`A.reshape((m,n))`	Reshape array A to array with same data (number of elements must match)
`A.resize((m,n))`	Resizes array A to $m \times n$ array
`A.flatten()`	Flattens array
`A.squeeze()`	Remove singe-dimensional entries from the shape of A
`A.repeat(n,2)`	Repeat array n times along dimension 3
`B = np.expand_dims(A,n)`	Expand the shape of an array to dimension n

1.2 Functions

1.2.1 Elementary Mathematical Functions

`y = np.sin(x)`	Trigonometric functions with argument x in radiant
`y = np.cos(x)`	
`y = np.tan(x)`	
`y = np.arcsin(x)`	Inverse trigonometric functions with return value y in radiant
`y = np.arccos(x)`	
`y = np.arctan(x)`	
`y = np.exp(x)`	Exponential function
`y = np.log(x)`	Natural logarithm
`y = np.log10(x)`	Logarithm with base 10
`y = np.sqrt(x)`	Square root
`y = np.abs(x)`	Absolute value
`y = np.round(x)`	Round to the next whole number
`y = np.floor(x)`	Round to the next smaller whole number
`y = np.ceil(x)`	Round to the next bigger whole number
`y = np.conj(x)`	Conjugate complex of x
`y = np.sum(x)`	Sum of all elements in array
`y = np.cumsum(x)`	Cummulative sum over all elements in array

1.2.2 Functions to calculate Characterisitic Values

`ma = x.max()`	Biggest value in an array
`mi = x.min()`	Smallest value in an array
`m = np.mean(x)`	Mean of all elements in an array
`s = np.std(x)`	Standard deviation of all elements in an array
`v = np.var(x)`	Variance of all elements in an array

1.3 Images

1.3.1 Spectrum

`X = np.fft.fft2(x)`	2D-Fast-Fourier-Transform
`X = np.fft.ifft2(x)`	Inverse 2D-Fast-Fourier-Transform

1.3.2 Filtering

To utilize signal processing, you have to import the `scipy` package!

`z = scipy.signal.convolve2d(x,y)`	2D-Convolution of x and y
`z = scipy.signal.correlate2d(x,y)`	2D-Corellation of x and y

1.4 Time measurement

To utilize time measurement, you have to import the `time` package!

`t = time.time()`	Get current time value

2 Matplotlib

For more see Matplotlib API
Prerequisites: `import matplotlib.pyplot as plt`

2.1 Graphical Functions

`fig = plt.figure(n)`	Makes figure n active or creates it, if it doesn't exist
`p = plt.subplot(m,n,i)`	Makes subplot active or creates it in current figure
`f,x = plt.subplots(m,n)`	Creates figure f and a set of $m \times n$ subplots x in f
`plt.plot(x,y)`	Plots y versus x as lines and/or markers
`plt.hist(x)`	Creates a histogramm plot
`plt.axis('off')`	Turn of axis lines and labels
`plt.axis([0,1,m,n])`	Makes plot axis from 0 to 1 in x-direction and m to n in y-direction
`plt.title('Text')`	Set title of active plot
`plt.xlabel('Text')`	Set label for the x-axis
`plt.ylabel('Text')`	Set label for the y-axis

2.2 Images

`f = plt.imread(path)`	Loads image file at location path as array
`plt.imsave(path,x)`	Save an array as image file to path
`plt.imshow(x)`	Display array as image

3 OpenCV

Be aware that OpenCV uses the BGR format for images, not RGB!
Prerequisites: `import cv2`

3.1 Load & Display Images

`img = imread(path)`	Loads image from a file
`imwrite(img,path)`	Saves image at path as a file
`imshow('name',img)`	Displays image in window with label 'name'

3.2 Video Capture

`c = cv2.VideoCapture(0)`	Start video capture sequence with video device 0
`c.isOpened()`	Returns if video capture is opened
`ret, frame = c.read()`	Grabs, decodes and returns the next video frame.

3.3 Image Manipulation

`g = filter2D(f,-1,r)`	Filters an image f with kernel r with the same depth as f

See more interesting image manipulation in OpenCVs Image Filtering API

3.4 Color Conversion

`g = cv2.cvtColor(f, cv2.COLOR_BGR2HSV)`	Converts BGR image to HSV
`g = cv2.cvtColor(f, cv2.COLOR_HSV2BGR)`	Converts HSV image to BGR

See more interesting color conversions in OpenCVs Miscellaneous Transformations API

4 Control Structures

4.1 `for` Loop

```
for variable in <list>:
     <statements>
```

```
for i in range(1, 11):
     y = y + x[i]
```

4.2 `while` Loop

```
while <expression>:
     <statements>
```

```
while i < 10:
      i = i*2
```

4.3 `if` Statement

```
if <expression>:
     <statements>
elif <expression>:
     <statements>
else:
     <statements>
```

```
if i >= 4:
     y = 1
elif i < -2:
     y = -1
else:
     y = 0
```

4.4 Self Defined Functions

```
def <name>(<inputs>):
     <statements>
```

```
def myMultiplication(a,b):
     return a*b
```