**Assignment 1.1 a)**

# return a square identity matrix of size dim+1

def identityMat(dim):

id\_mat = ()

for i in range (0, dim+1):

id\_mat += ((0,)\*i+(1,)+(0,)\*(dim-i),)

for i in range (0, dim+1):

print id\_mat[i]

return id\_mat

Eingabe:

identityMat(3)

Ausgabe:

(1, 0, 0, 0)

(0, 1, 0, 0)

(0, 0, 1, 0)

(0, 0, 0, 1)

((1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1))

**Assignment 1.1 b)**

# return a quadratic scaling matrix of size len(s)+1

# the values in the tuple 's' represent the scale factors

# of the respective diagonal elements

def scaleMat(s):

dim = len(s)

sc\_mat = ()

sl = list(s)

sl.append(1)

# concatenate tuples

for i in range (0,dim+1):

sc\_mat += ((0,)\*i+(sl[i],)+(0,)\*(dim-i),)

for i in range (0,dim+1):

print sc\_mat[i]

return tuple(sc\_mat)

Eingabe:

s = (3, 1, 4)

scaleMat(s)

Ausgabe:

(3, 0, 0, 0)

(0, 1, 0, 0)

(0, 0, 4, 0)

(0, 0, 0, 1)

((3, 0, 0, 0), (0, 1, 0, 0), (0, 0, 4, 0), (0, 0, 0, 1))

**Assignment 1.1 c)**

# return a quadratic translation matrix

# that, when applied to a point, moves it by 't'

def translationMat(t):

dim = len(t)

matrix = ()

sl = list(t)

for i in range(0, dim):

matrix += ((0,)\*i+(1,)+(0,)\*(dim-i-1)+(sl[i],),)

matrix += ((0,)\*dim+(1,),)

for i in range(0, dim+1):

print matrix[i]

return matrix

Eingabe:

t = (4,2)

translationMat(t)

Ausgabe:  
(1, 0, 4)

(0, 1, 2)

(0, 0, 1)

((1, 0, 4), (0, 1, 2), (0, 0, 1))

**Funktion multiply: multipliziert zwei Matrizen**

# Matrix multiplication

# Matrices are two-dimensional tuples

def multiply( m0, m1 ):

try:

# multiplication by scalar

float(m1)

return tuple([ tuple([ e\*m1 for e in r ]) for r in m0 ])

except :

pass

#convert vector to matrix

convert = ()

if not isinstance(m1[0],tuple):

for i in range(0,len(m1)):

convert += ((m1[i],),)

m1 = convert

if len(m1) == len(m0[0])-1:

m1 +=((1,),)

# multiply matrices m0 x m1

result = ()

for i in range(0,len(m0)):

rTmp = ()

for j in range(0,len(m1[0])):

val = 0;

for k in range(0,len(m1)):

m0\_ = m0[i][k]

m1\_ = m1[k][j]

val += m0\_\*m1\_

rTmp += (val,)

result += (rTmp,)

# convert one-colmun matrices back to vector (flat tuple)

if len(result[0]) == 1:

convert = ()

for i in range(0,len(result)):

convert += (result[i][0],)

result = convert

return result

**Assignment 2.3**

##Function to adress a 2-dimensional matrix

##using a 1-dimensional index.

##Function should return the value (1 number!) of the

##2-dimensional matrix using the 1-dimensional index

##one whould use, if the matrix was linearized row wise

def addressMatrixWith1dIndex(matrix, index, nrow, ncol):

# global index: index = j \* ncol + i

# indices: j = index // ncol, integer division

# i = index % ncol

val = matrix[index//ncol][index%ncol]

return val

##Takes a 2dimensional matrix (represented by a list of lists)

##as input, linearizes it row wise and returns the resulting

##1d list

def convert2dTo1d(matrix, nrow, ncol):

array = []

[array.append([x for x in row]) for row in matrix]

return array

Eingabe:

matrix = ((3,4,2),(2,4,2),(3,5,3))

convert2dTo1d(matrix, 3, 3)

Ausgabe:

[[3, 4, 2], [2, 4, 2], [3, 5, 3]]

##Takes a 1dimensional list as input, that is the result of

##a row wise linearized matrix

##and returns the 2dimensional matrix as a list of lists

def convert1dTo2d(array, nrow, ncol):

matrix = []

for i in range(nrow):

row = []

for j in range(ncol):

row.append(array[i\*ncol + j])

matrix.append(row)

return matrix

Eingabe:

array = (3,6,1,5,6,3)

convert1dTo2d(array, 3, 2)

Ausgabe:

[[3, 6], [1, 5], [6, 3]]

##Takes a images stored in a 1dimensional data array

##and prints it in form of a 2dimensional matrix

##Use sys.stdout.write("To print") instead of

##print "To print" to avoid unwanted whitespaces and

##newlines

def printImage(array, nrow, ncol):

for i in range(nrow):

for j in range(ncol):

sys.stdout.write(array[i\*ncol+j])

return