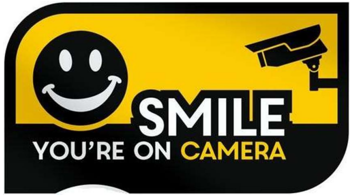
KOÇ ÜNİVERSİTESİ

Lab10

COMP 125 Programming with Python

Recording Disclaimer



The synchronous sessions are recorded (audiovisual recordings). The students are not required to keep their cameras on during class.

The audiovisual recordings, presentations, readings and any other works offered as the course materials aim to support remote and online learning. They are only for the personal use of the students. Further use of course materials other than the personal and educational purposes as defined in this disclaimer, such as making copies, reproductions, replications, submission and sharing on different platforms including the digital ones or commercial usages are strictly prohibited and illegal.

The persons violating the above-mentioned prohibitions can be subject to the administrative, civil, and criminal sanctions under the Law on Higher Education Nr. 2547, the By-Law on Disciplinary Matters of Higher Education Students, the Law on Intellectual Property Nr. 5846, the Criminal Law Nr. 5237, the Law on Obligations Nr. 6098, and any other relevant legislation.

The academic expressions, views, and discussions in the course materials including the audio-visual recordings fall within the scope of the freedom of science and art.

Matrix Operations

A matrix is a collection of n-by-m elements in n-rows and m-columns with a_{ij} denoting the element of the matrix at the i-th row and j-th column . Matrices can be stored as list-of-lists in python

$$A=\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2m} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3m} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix}$$

```python
A=[ [1, 2, 3], [3, 2, 1]]
```

Matrix Sum

Summation of two matrices A and B with identical dimensions

(same number of rows and columns) is performed as follows:

$$
\begin{bmatrix}
a_{11} & a_{12} & \dots & a_{1m} \\
a_{21} & a_{22} & \dots & a_{2m} \\
\dots & \dots. & \dots. & \dots \\
a_{n1} & a_{n2} & \dots & a_{nm}
\end{bmatrix}
+
\begin{bmatrix}
b_{11} & b_{12} & \dots & b_{1m} \\
b_{21} & b_{22} & \dots & b_{2m} \\
\dots & \dots & \dots & \dots \\
b_{n1} & b_{n2} & \dots & b_{nm}
\end{bmatrix}
=
\begin{bmatrix}
a_{11}+b_{11} & a_{12}+b_{12} & \dots & a_{1m}+b_{1m} \\
a_{21}+b_{21} & a_{22}+b_{22} & \dots & a_{2m}+b_{2m} \\
\dots & \dots & \dots & \dots \\
a_{n1}+b_{n1} & a_{n2}+b_{n2} & \dots & a_{nm}+b_{nm}
\end{bmatrix}
$$

Matrix functions

Download the file Lab10a.py

The file contains three matrix functions that are already implemented

```
import random

def matrix_generate(n,m):
    """Receives the number of rows n and number of columns m
    and generates a matrix of size n-by-m with entries choosen as
    random values in the range (0,10).
    The return matrix is stored as a list-of-lists"""

    a = [[ random.randint(0,10) for i in range(m)] for j in range(n)]

    return a

def matrix_check(a):
    """Given a matrix a check if each row has the same elements.
    If corrent return True, else return False."""

    n=len(a)
    m=len(a[0])
    for i in range(1,n):
        if len(a[i]) != m:
            return False
    return True

def matrix_display(a):
    """Display the entries of a matrix as shown in the slides
    This function has no return. It simply prints a
    formatted view of the matrix. """

    n=len(a)
    m=len(a[0])
    out=""
    for i in range(n):
        out += "["
        for j in range(m):
            out += f"{a[i][j]:4}"
        out +=" ]\n"
    print(out)
```

Matrix functions

Your task is to implement

```
def matrix_sum(a,b):
    """ Given two matrices a and b, check the size
    of the matrices.
    If the sizes match, calculate the sum of the matrices
    and return the resulting matrix.
    """

    return
```

It should work as follows:

```
In [2]: #Let's create two 2-by-3 matrices
   ...: n = 2
   ...: m = 3
   ...: A = matrix_generate(n,m)
   ...: B = matrix_generate(n,m)

In [3]: #Display the matrices
   ...: print("Matrix A")
   ...: matrix_display(A)
   ...: print("Matrix B")
   ...: matrix_display(B)
Matrix A
[   4   7   0 ]
[   8   3   9 ]

Matrix B
[   1   1  10 ]
[   4   9   1 ]
```

```
In [4]: #Check the size of the matrices
   ...: print("Is A a proper matrix?")
   ...: print(matrix_check(A))
   ...: print("Is B a proper matrix?")
   ...: print(matrix_check(B))
Is A a proper matrix?
True
Is B a proper matrix?
True

In [5]: #Display the sum of the matrices
   ...: print("Matrix C = A+B")
   ...: C = matrix_sum(A,B)
   ...: matrix_display(C)
Matrix C = A+B
[   5   8  10 ]
[  12  12  10 ]
```

Sparse Matrix

- A matrix is called a Sparse Matrix if majority of its elements are zero.
- Storing a sparse matrix as list-of-lists is not very efficient.
- Alternatively, one can store only the non-zero elements, by using a dictionary in Python.
- **keys**: indices (i,j) of nonzero elements
- **values**: non-zero values to be stored.
- In order to keep the size of the matrix, add a special key **'size'**, which contains a tuple **(n,m)** containing the matrix dimensions.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Sparse Matrix functions

Download the file Lab10b.py

The file contains two functions that are already implemented.

It also imports Lab10a.py as a module. So this file has to be in the same folder.

```
import Lab10a
import random
def generate_random_sparse_matrix(n, m, spar=0.9):
    """
    This function generates a random matrix as a
    list of lists with a given sparsity.
    """

    a = [[ 0 for i in range(m)] for j in range(n)]

    #How many elements should be nonzero
    nr = int((1.-spar)*n*m)
    pos = random.sample(range(n*m), nr)
    for ind in pos:
        n_ind = int(ind/n)
        m_ind = ind-n_ind*m
        a[n_ind][m_ind]=1
    return a

def dense_to_sparse(A):
    """
    This function converts a list-of-lists type matrix
    to the sparse representation stored in a dictionary
    """
    sp = {}

    n = len(A)
    m = len(A[0])

    for i in range(n):
        for j in range(m):
            if A[i][j]:
                sp[(i,j)]=A[i][j]
    sp['size']= n,m
    return sp
```

Sparse Matrix functions

Your task is to implement

```python
def matrix_add_sparse(A, B):
    """
    Given two dictionaries, which contain the sparse matrix
    representations of matrices A and B, this function calculates the
    matrix sum and returns a new dictionary.

    """
```

Hint: You should not write a nested for-loop structure to iterate over the full size matrix. This would defeat the purpose !!!

Sparse Matrix functions

It should work as follows

```
In [31]: #Let's generate a sparse matrix of size 4-by-4
    ...: #Only half the elements are non-zero
    ...: A = (generate_random_sparse_matrix(4,4,.5))
    ...: matrix_display(A)
[   1   1   0   0 ]
[   0   1   0   0 ]
[   1   0   1   0 ]
[   0   1   1   1 ]


In [32]: #Convert list-of-lists to sparse representation
    ...: A_sp = dense_to_sparse(A)
    ...: print(A_sp)
{(0, 0): 1, (0, 1): 1, (1, 1): 1, (2, 0): 1, (2, 2): 1, (3, 1): 1, (3, 2): 1, (3, 3): 1, 'size': (4, 4)}

In [33]: #Create a second matrix
    ...: B = (generate_random_sparse_matrix(4,4,.5))
    ...: matrix_display(B)
    ...:
    ...: #Convert list-of-lists to sparse representation
    ...: B_sp = dense_to_sparse(B)
    ...: print(B_sp)
[   1   0   0   0 ]
[   0   0   1   1 ]
[   1   1   1   0 ]
[   0   0   1   1 ]

{(0, 0): 1, (1, 2): 1, (1, 3): 1, (2, 0): 1, (2, 1): 1, (2, 2): 1, (3, 2): 1, (3, 3): 1, 'size': (4, 4)}

In [34]: #Here is the sum of matrices A+B
    ...: print(matrix_add_sparse(A_sp,B_sp))
{(0, 0): 2, (0, 1): 1, (1, 1): 1, (2, 0): 2, (2, 2): 2, (3, 1): 1, (3, 2): 2, (3, 3): 2, 'size': (4, 4), (1, 2): 1,
(1, 3): 1, (2, 1): 1}
```