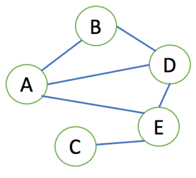
Assignment #4

In this assignment, we will practice on NumPy, Tuple, Set, Dict using social network use case.



(A)

Adjacency Matrix A

	A	B	C	D	E
A	0	1	0	1	1
B	1	0	0	1	0
C	0	0	0	0	1
D	1	1	0	0	1
E	1	0	1	1	0

(B)

Figure 1: (A) social network (B) adjacency matrix of this social network

Figure 1A shows an example of a social network graph indicating that

A has B, D, and E as friends,

B has A and D as friends,

C has only E as friend,

D has A, B, and E as friends, and

E has A, C, and D as friends.

From this graph, we can transform it into an adjacency matrix (A) as shown in Figure 1B.

Task#1

Write a function **`generate_adjacency_matrix(filename)`** where filename is the name of the input file "social_network.txt" and each row within the file indicates that the two persons are friends.

Note: assume that person names are unique.

"`social_network.txt`"

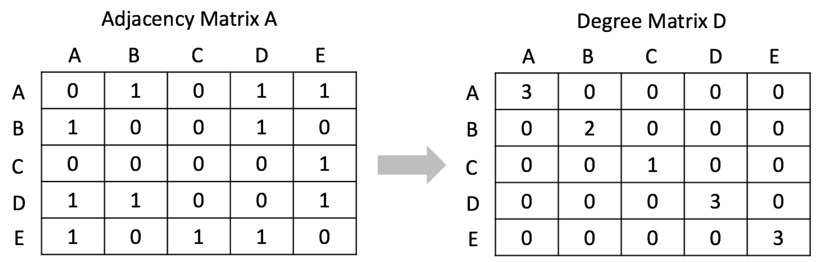
```
A,B
A,D
A,E
B,A
B,D
C,E
D,A
D,B
D,E
E,A
E,C
E,D
```

The output of this function is (1) a numpy array representing the adjacency matrix A of the input social network, (2) a list of person names in the same order as of their rows in the adjacency matrix A. Hint: use data type int for generating the adjacency matrix A.

Example of expected output.

```
[[0 1 0 1 1]
 [1 0 0 1 0]
 [0 0 0 0 1]
 [1 1 0 0 1]
 [1 0 1 1 0]]
['A', 'B', 'C', 'D', 'E']
```

After getting the adjacency matrix A, we want to transform this matrix into a degree matrix as shown in Figure 2.



Adjacency Matrix A

	A	B	C	D	E
A	0	1	0	1	1
B	1	0	0	1	0
C	0	0	0	0	1
D	1	1	0	0	1
E	1	0	1	1	0

Degree Matrix D

	A	B	C	D	E
A	3	0	0	0	0
B	0	2	0	0	0
C	0	0	1	0	0
D	0	0	0	3	0
E	0	0	0	0	3

Figure 2: Adjacency matrix and degree matrix

This degree matrix D always has the same dimensions as of the adjacency matrix A and the diagonal of the degree matrix will contain the sum of all friends of each person.

Task#2

Write a function **`get_degree_matrix(A)`**, where the input of this function is the adjacency matrix A from Task#1 and the output of this function is the degree matrix D as shown below.

Example of expected output.

```
[[3 0 0 0 0]
 [0 2 0 0 0]
 [0 0 1 0 0]
 [0 0 0 3 0]
 [0 0 0 0 3]]
```

After getting the degree matrix D, we would like to use this matrix to find the names (e.g, A, B, ..) of all persons who have the highest number of friends.

Task#3

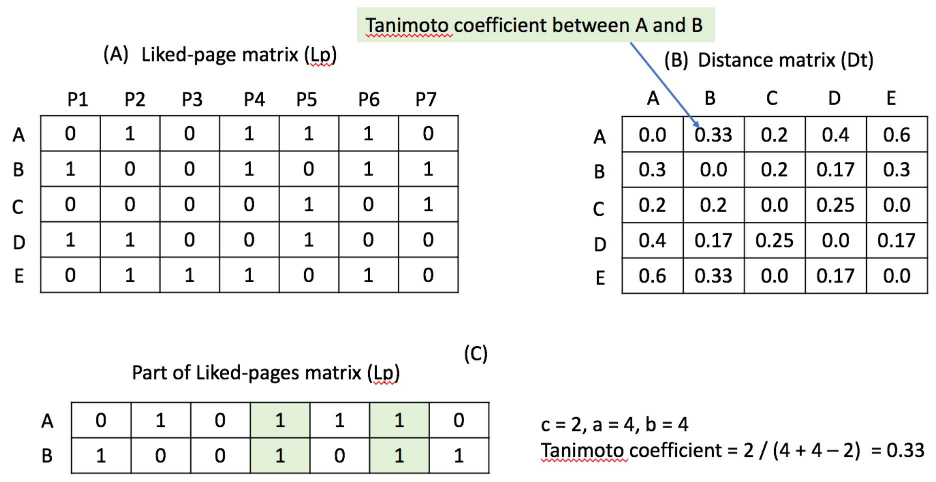
Write a function **`get_names_with_highest_number_of_friends(D,person_names)`**, where the input of this function is the degree matrix D from Task#2 and the list of person names from Task#1.

The output of this function is the list of person names who have the highest number of friends and the indexed rows of these persons in the degree matrix D.

Example of expected output.

```
[('A', 0), ('D', 3), ('E', 4)]
```

One month later, we get more information about all persons' liked pages and this information is in binary bits as shown in Figure 3A. For examples, A likes pages P2, P4, P5, and P6. Figure 3B shows an example of the distance matrix (Dt) calculated from the liked-page matrix.


Tanimoto coefficient between A and B
(A) Liked-page matrix (Lp)
(B) Distance matrix (Dt)
(C)
Part of Liked-pages matrix (Lp)
c = 2, a = 4, b = 4
Tanimoto coefficient = 2 / (4 + 4 – 2) = 0.33

	P1	P2	P3	P4	P5	P6	P7
A	0	1	0	1	1	1	0
B	1	0	0	1	0	1	1
C	0	0	0	0	1	0	1
D	1	1	0	0	1	0	0
E	0	1	1	1	0	1	0

	A	B	C	D	E
A	0.0	0.33	0.2	0.4	0.6
B	0.3	0.0	0.2	0.17	0.3
C	0.2	0.2	0.0	0.25	0.0
D	0.4	0.17	0.25	0.0	0.17
E	0.6	0.33	0.0	0.17	0.0

A	0	1	0	1	1	1	0
B	1	0	0	1	0	1	1

Figure 3 Example of how to calculate distance matrix (Dt) from a feature matrix, i.e., liked-page matrix, based on Tanimoto coefficient.

Each cell of the Dt contains the Tanimoto coefficient calculated by the following equation.

Tanimoto (Jaccard) coefficient = c / (a + b – c)

where c is the number of pages liked by both A and B and a is the number of pages liked by A while b is the number of pages liked by B.

The function to get the liked-page matrix (Lp) and person names in the same order of the data in the matrix is provided as get_liked_page_matrix("liked_pages.txt") and its output is shown below.

Example of liked page matrix

```
[[0 1 0 1 1 1 0]
 [1 0 0 1 0 1 1]
 [0 0 0 0 1 0 1]
 [1 1 0 0 1 0 0]
 [0 1 1 1 0 1 0]]
['A', 'B', 'C', 'D', 'E']
```

Task#4

Write a function **`get_distance_matrix(Lp)`**, where input of the function is the liked-page matrix (Lp) and the output will be the distance matrix (Dt) as shown below.

Example of expected output.

```
[[0.         0.33333333 0.2        0.4        0.6       ]
 [0.33333333 0.         0.2        0.16666667 0.33333333]
 [0.2        0.2        0.         0.25       0.        ]
 [0.4        0.16666667 0.25       0.         0.16666667]
 [0.6        0.33333333 0.         0.16666667 0.        ]]
```

Task#5

Based on the distance matrix (Dt), write a function **`get_most_similar_pairs(Dt, persons)`**, where input of the function is the distance matrix (Dt) and a list of person names from get_liked_page_matrix(), and the output will be a list of tuples where each tuple represents a pair of person names (alphabetically ordered), who are most similar based on their liked pages. Only pairs with the same highest Tanimoto coefficient will be returned.

Example of expected output.

```
[('A', 'E')]
```