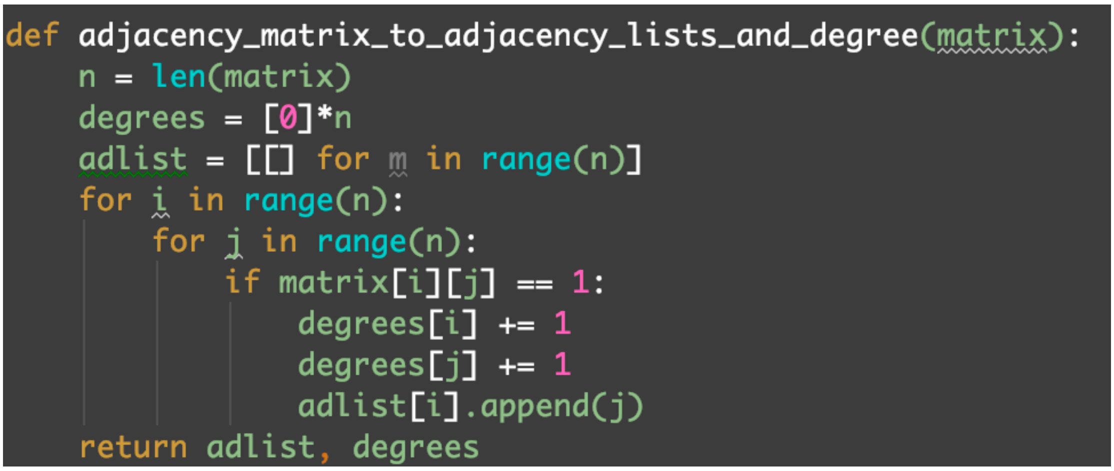
Lab 13 Report

1. Lab exercises

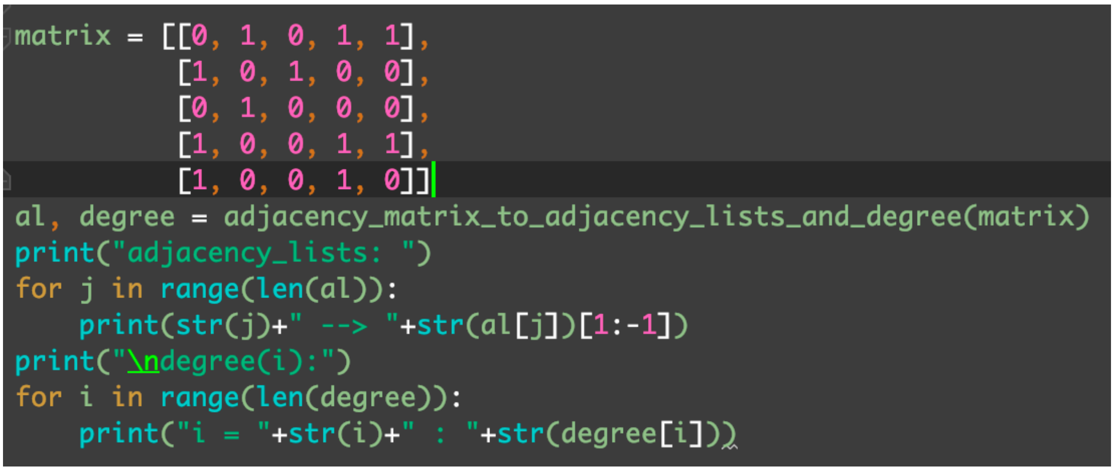
1.1 Given an undirected Graph (V, E), in Adjacency Matrix, generate its Adjacency Lists, using it to calculate and print out degree(i)

To generate the adjacency lists with adjacency matrix, we need to get every “j” for each “i” such “adjacency matrix[i][j]” equal to the 1 and store the “j” for a single “i” into a list and use these lists to generate a double list, which is the adjacency lists. And the length of “adjacency lists[i]” is the “degree[i]”.



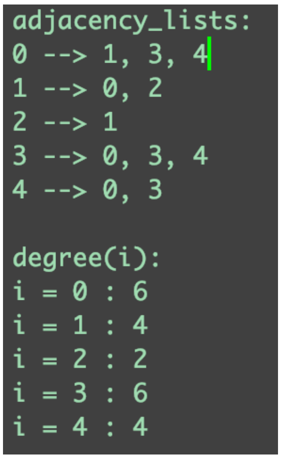
```python
def adjacency_matrix_to_adjacency_lists_and_degree(matrix):
    n = len(matrix)
    degrees = [0]*n
    adlist = [[] for m in range(n)]
    for i in range(n):
        for j in range(n):
            if matrix[i][j] == 1:
                degrees[i] += 1
                degrees[j] += 1
                adlist[i].append(j)
    return adlist, degrees
```

Test:



```python
matrix = [[0, 1, 0, 1, 1],
          [1, 0, 1, 0, 0],
          [0, 1, 0, 0, 0],
          [1, 0, 0, 1, 1],
          [1, 0, 0, 1, 0]]
al, degree = adjacency_matrix_to_adjacency_lists_and_degree(matrix)
print("adjacency_lists: ")
for j in range(len(al)):
    print(str(j)+" --> "+str(al[j])[1:-1])
print("\ndegree(i):")
for i in range(len(degree)):
    print("i = "+str(i)+" : "+str(degree[i]))
```

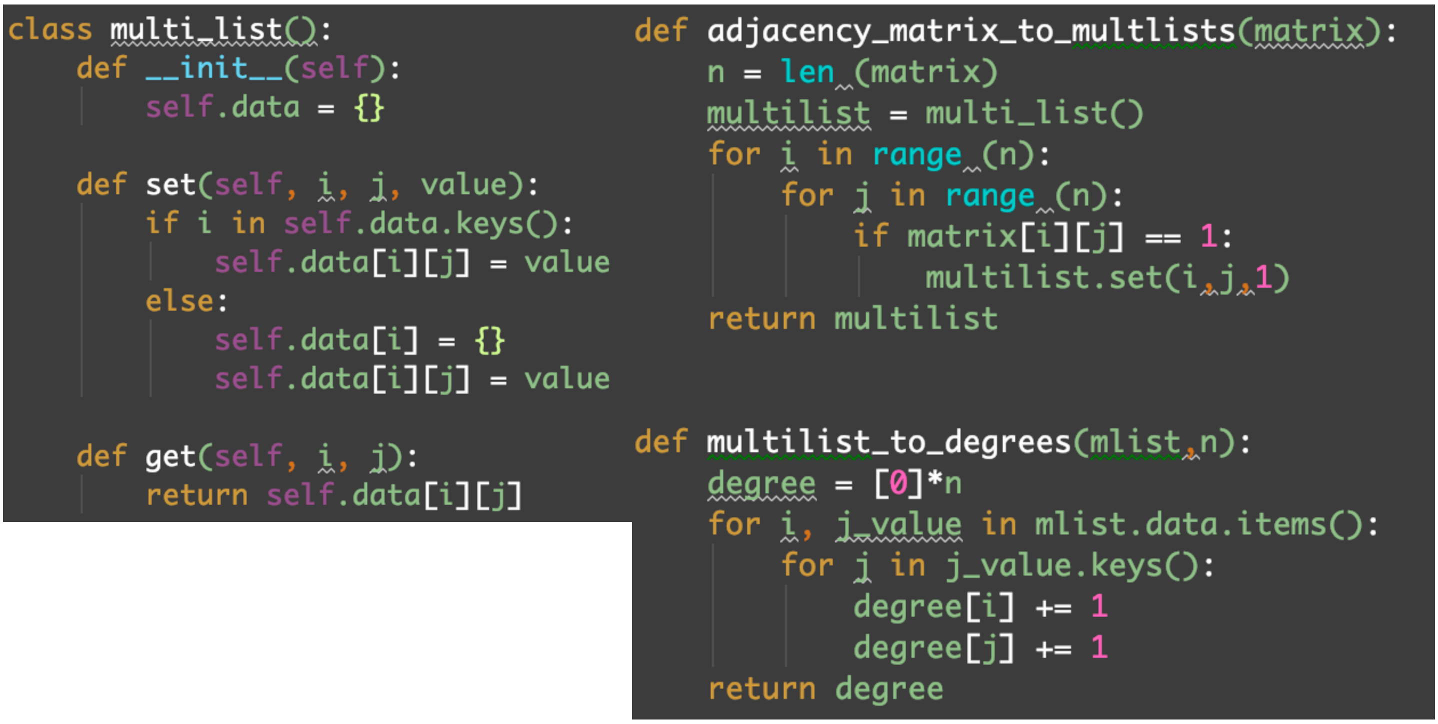
Output:



```
adjacency_lists:
0 --> 1, 3, 4
1 --> 0, 2
2 --> 1
3 --> 0, 3, 4
4 --> 0, 3

degree(i):
i = 0 : 6
i = 1 : 4
i = 2 : 2
i = 3 : 6
i = 4 : 4
```

1.2 Given a directed Graph (V, E), in Adjacency Matrix, generate its MultLists, using it to calculate and print out degree(i).

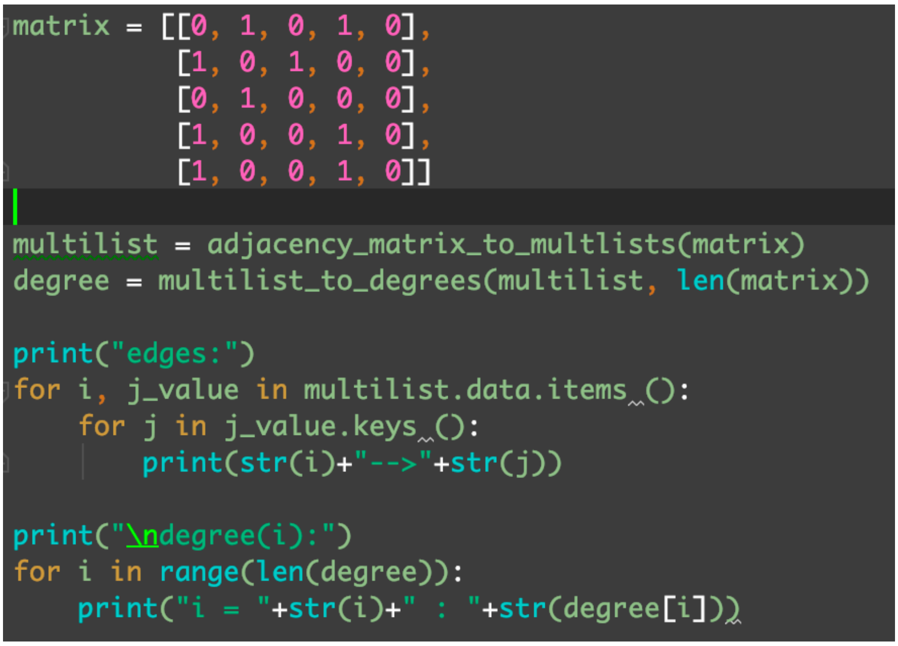
To generate the multlists with adjacency matrix, we need to get every “j” for each “i” such “adjacency matrix[i][j]” equal to the 1 and store the “j” and “i” into the multlists. Then, for every single “i”, the number of relevant “j” in the mutilists is the “degree[i]”.



```
class multi_list():
    def __init__(self):
        self.data = {}

    def set(self, i, j, value):
        if i in self.data.keys():
            self.data[i][j] = value
        else:
            self.data[i] = {}
            self.data[i][j] = value

    def get(self, i, j):
        return self.data[i][j]
```

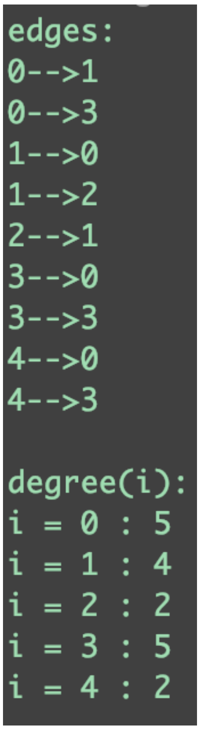
```
def adjacency_matrix_to_multlists(matrix):
    n = len (matrix)
    multilist = multi_list()
    for i in range (n):
        for j in range (n):
            if matrix[i][j] == 1:
                multilist.set(i,j,1)
    return multilist


def multilist_to_degrees(mlist,n):
    degree = [0]*n
    for i, j_value in mlist.data.items():
        for j in j_value.keys():
            degree[i] += 1
            degree[j] += 1
    return degree
```

Test:



```
matrix = [[0, 1, 0, 1, 0],
          [1, 0, 1, 0, 0],
          [0, 1, 0, 0, 0],
          [1, 0, 0, 1, 0],
          [1, 0, 0, 1, 0]]

multilist = adjacency_matrix_to_multlists(matrix)
degree = multilist_to_degrees(multilist, len(matrix))

print("edges:")
for i, j_value in multilist.data.items():
    for j in j_value.keys():
        print(str(i)+"-->"+str(j))

print("\ndegree(i):")
for i in range(len(degree)):
    print("i = "+str(i)+" : "+str(degree[i]))
```

Output:

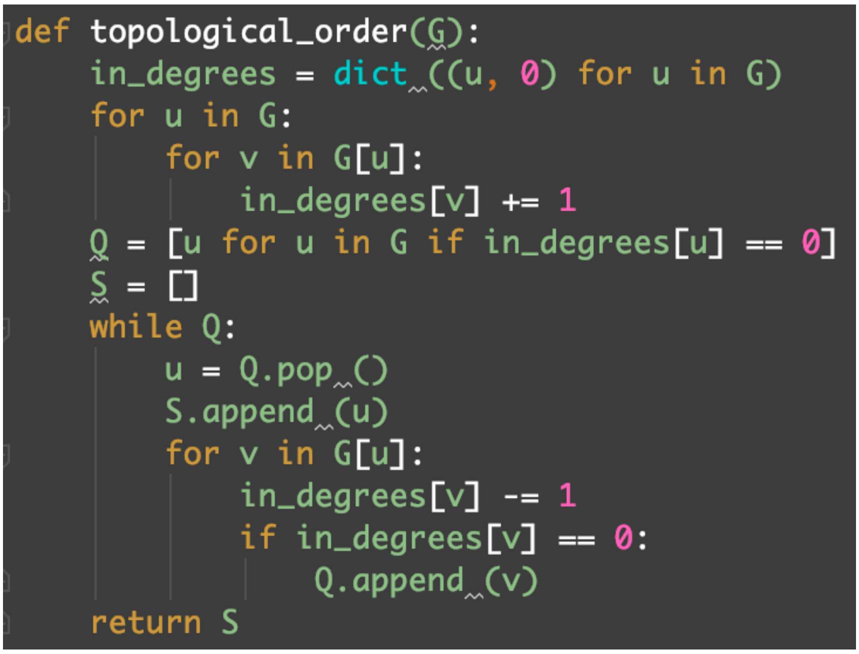


```
edges:
0-->1
0-->3
1-->0
1-->2
2-->1
3-->0
3-->3
4-->0
4-->3

degree(i):
i = 0 : 5
i = 1 : 4
i = 2 : 2
i = 3 : 5
i = 4 : 2
```

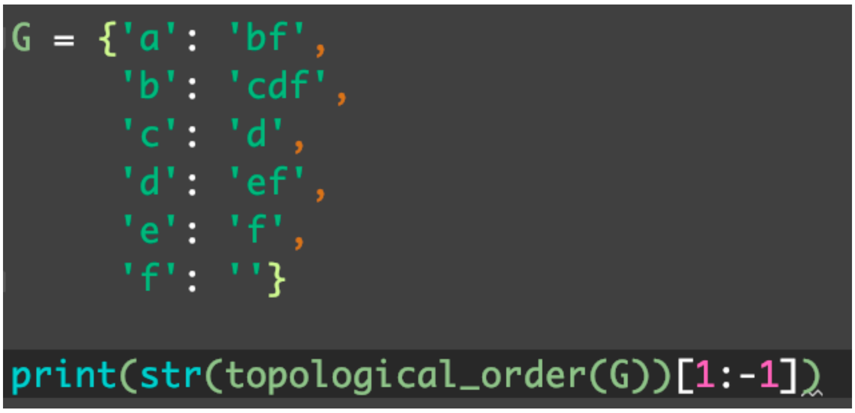
1.3 Given an AOV network, generate and print a topological order if possible.

Remove the node with 0 indegrees form the node and reset indegrees of the nodes that relevant to the removed node. Repeat the operation until all of the nodes are removed from the graph.

Test:

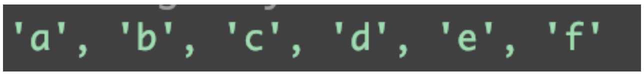


```
def topological_order(G):
    in_degrees = dict ((u, 0) for u in G)
    for u in G:
        for v in G[u]:
            in_degrees[v] += 1
    Q = [u for u in G if in_degrees[u] == 0]
    S = []
    while Q:
        u = Q.pop ()
        S.append (u)
        for v in G[u]:
            in_degrees[v] -= 1
            if in_degrees[v] == 0:
                Q.append (v)
    return S
```



```
G = {'a': 'bf',
     'b': 'cdf',
     'c': 'd',
     'd': 'ef',
     'e': 'f',
     'f': ''}

print(str(topological_order(G))[1:-1])
```

Output:



```
'a', 'b', 'c', 'd', 'e', 'f'
```