Çelik Köseoğlu 21400196 - CS315 Section 2

Part A)

1. otherData = data # same for both built-in and NumPy arrays

   This assignment will bind otherData into data. Python interpreter will only copy the reference to the object. This is called shallow copying.

   In this case, Python interpreter point both “data” and “otherData” to the same location in memory. So, when you modify the value using the “otherData” reference, you will end up modifying the location which “data” points to as well.

2. otherData = data[1:3] # different for built-in and NumPy arrays

   In built-in python lists, this operation will result in a deep copy. A new array will be created with size (3-1 = 2). This array will copy the elements from the provided range of the other array.

   However, NumPy will not create a new array and copy the objects. It will only reference the objects in the range (aka shallow copy). So, when you modify the referenced memory location, you modify both array’s contents.

Part B)

In this part, I will be listing the differences and similarities, and explaining them.

Similar:

When we want to refer to a specific index element, the code is similar. Let’s say that we have a [5][5] matrix and want to refer to index [1][3].

In NumPy: matrix[1,3] will return the element
In built-in arrays: matrix[1][3] will return the element

Different:

Built-in arrays handle matrix operations like regular multiplication on normal numbers. For example, it will multiply the [1][2] element on matrix M with [1][2] on matrix N. Then it will write this result in the [1][2] index of the resulting matrix.

In NumPy, multiplication on matrices will be like regular matrix multiplication in Linear Algebra. Operations like M * N and M ** N will be carried out according to matrix multiplication rules.

Part C)

matrixConverter.py file is included in the directory

Part D)

A sparse matrix is a matrix which most of its elements are zero. In our case, the adjacency matrix we're storing is a sparse matrix.

There is a library called Scipy which handles sparse matrices with much less memory than NumPy. It only stores elements different than zero, so you won't have to allocate space for zeros. In our case, it would be very beneficial to use since the matrix in the example input has 14 zeros and 11 ones. We would have saved more than 50% memory in we had used Scipy.

Moreover, it is very easy to define Scipy sparse matrices:

```
>>>A = csr_matrix([[1, 2, 0], [0, 0, 3], [4, 0, 5]])

# taken from https://docs.scipy.org/doc/scipy-0.18.1/reference/sparse.html
```

There are many other libraries which handle sparse matrices. One other example is the igraph library. However, Scipy is the most recent and the easiest one to use for now.

Part E)

Account.py file is included in the directory with createNameMap function