Mert Aytöre - 21400293

CS315-02 – Homework 1

5 March 2016

In the first two parts of this homework, we examine two implementations of sequence of numbers in the language Python. Starting from the first line, in built-in Python lists, the initialization with a single line is sufficient. In NumPy arrays, to work with them, we first need to import numpy and state the way that we are going to use it. In the homework, it is stated as “np”. For initialization, variable name stays the same but we’re specifying that numpy’s array will be used with the line np.array([1, 2, 3, 4]). Printing the contents of the list & array is done by the same line, print data. The purpose of the code is the same but the output has a small difference. They both start with an opening square bracket, end with a closing bracket and consist of the values inside them. However, built-in lists separates the values with a comma and a space after that (‘, ’) and NumPy arrays separates the values only with a single space.

Part A)

In this part, there are two things to consider as similarities and differences. The operations are mainly assignment operations but there are small differences that distinguishes the results of the operations. When the initialization otherData = data is made, otherData functions something like a pointer. Since they are different names for a single list, variables data and otherData control the same array if any change is made on them. It can be easily seen by printing the arrays after the changes. Same applies for NumPy arrays.

The other thing to consider is that their differences as in initialization subarrays.

In built-in lists with this line of code; otherData = data[1:3], otherData is given the values between 1st and the 3rd indexes in data list and behaves as a new list. It is now separated from data list.

The same initialization is slightly different in NumPy arrays. When executing this line of code; otherData = data[1:3], otherData is given the values between 1st and the 3rd indexes in data list. However, when printing data array, the changes made on otherData will affect the indexes between 1 and 3 because otherData behaves as if it’s a pointer to the values between the indexes specified.

Mert Aytöre - 21400293

CS315-02 – Homework 1

5 March 2016

Part B)

In this part, we are comparing built-in 2D arrays and NumPy matrices. They both result in the same way when we want to get a specific item from the list/matrix. However, they differ with the multiplication operation in the given code blocks.

Built-in 2D arrays handles the multiplication operation by multiplying the values in the matching indexes. To be more precise, after these two lines are executed;

```
>>> A = np.array([[1,2], [3,4]])
>>> B = np.array([[2,1], [-1,2]])
```

the operation flow is in the following:

1*2, insert the value into the as the first element of the resulting 2D array.

2*1, insert the value into the as the second element of the resulting 2D array.

3*-1, insert the value into the as the third element of the resulting 2D array.

4*2, insert the value into the as the fourth element of the resulting 2D array.

On the other hand, NumPy multiplies the matrices with the proper matrix multiplication in Linear Algebra. Thus the results coming from the operations A *B and A**3 are obtained accordingly.

Part D)

On a large scale, matrix representation will require space which is more than needed. Also doing operations on the matrix will waste a lot of time because unwanted zeros will always be visited while perform them. To avoid having such problems, adjacency lists can be considered as a solution to represent only the used terms (non-zero terms in matrix) in the graph.

Implementation of adjacency lists requires an array of linked lists. The size of the array will be the number of total vertices. The edges are represented as the nodes connected to the each head in the array. The nodes represent the whether the vertices are connected via edges to each other or not.

There exists a Python library called Another Python Graphical Library (APGL). There is also a SparseGraph implementation which exactly does what we want. Example code block for a graph representation is in the following:

Mert Aytöre - 21400293

CS315-02 – Homework 1

5 March 2016

```
from apgl.graph import SparseGraph
import numpy

numVertices = 10

graph = SparseGraph(numVertices)
graph.addEdge(0, 1)
#another notation
graph[0, 2] = 1
graph[0, 3] = 1
graph[2, 1] = 1
graph[2, 5] = 1
graph[2, 6] = 1
graph[6, 9] = 1
```

There are also other libraries which does the same thing and they are NetworkX and igraph. They both have SparseGraph implementations which use scipy, numpy and matlib as well.