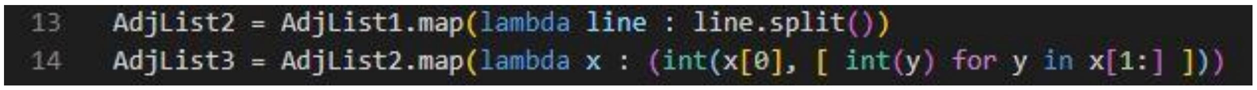
1) Explanation of the source code:

What are the functions of your lambda functions? Which kind of intermediate results are generated?

These are the lambda functions:

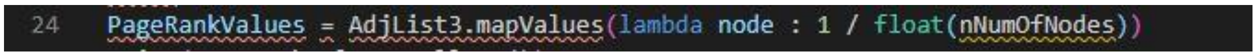
For the first one on line 13, we are splitting the line based off the space because we need each of the strings to be separate words.

For the second one on line 14, lambda function is mapping it to the first element of the tuple which is 0 and for each element in the list we want to save it as a list.



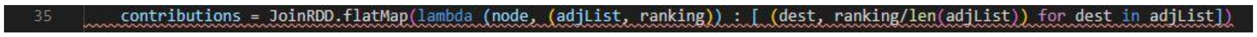
```
13    AdjList2 = AdjList1.map(lambda line : line.split())
14    AdjList3 = AdjList2.map(lambda x : (int(x[0], [ int(y) for y in x[1:] ]))
```

For the third one on line 24, lambda function is everything that is mapValue represented by 1 divided by the total number of nodes.



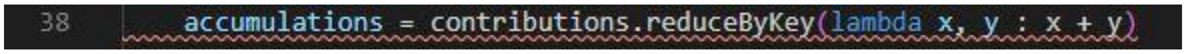
```
24    PageRankValues = AdjList3.mapValues(lambda node : 1 / float(nNumOfNodes))
```

For the forth one on line 35, lambda function divided each rank by the values of the adjacency list. The lambda passes the tuple value x, y, and z. For every element in the tuple it will be mapping it to the likelihood that individual element is the one we go to.

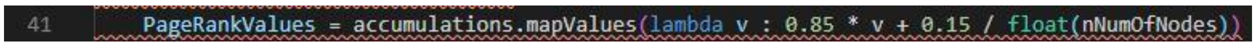


```
35        contributions = JoinRDD.flatMap(lambda (node, (adjList, ranking)) : [ (dest, ranking/len(adjList)) for dest in adjList])
```

For the fifth one on line 38, the lambda function grabs its accumulations and adds x and y to get the total level of accumulations.



```
38        accumulations = contributions.reduceByKey(lambda x, y : x + y)
```



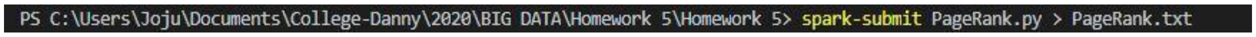
```
41        PageRankValues = accumulations.mapValues(lambda v : 0.85 * v + 0.15 / float(nNumOfNodes))
```

For the sixth one on line 41, the lambda function recalculates the number of nodes as a floating point. Each v is the total evaluation that we got off the contributions. We are resetting the page rank values for each individual node. Recalculating all of them based off their contributions and reassessing their values.

2) Experimental Results

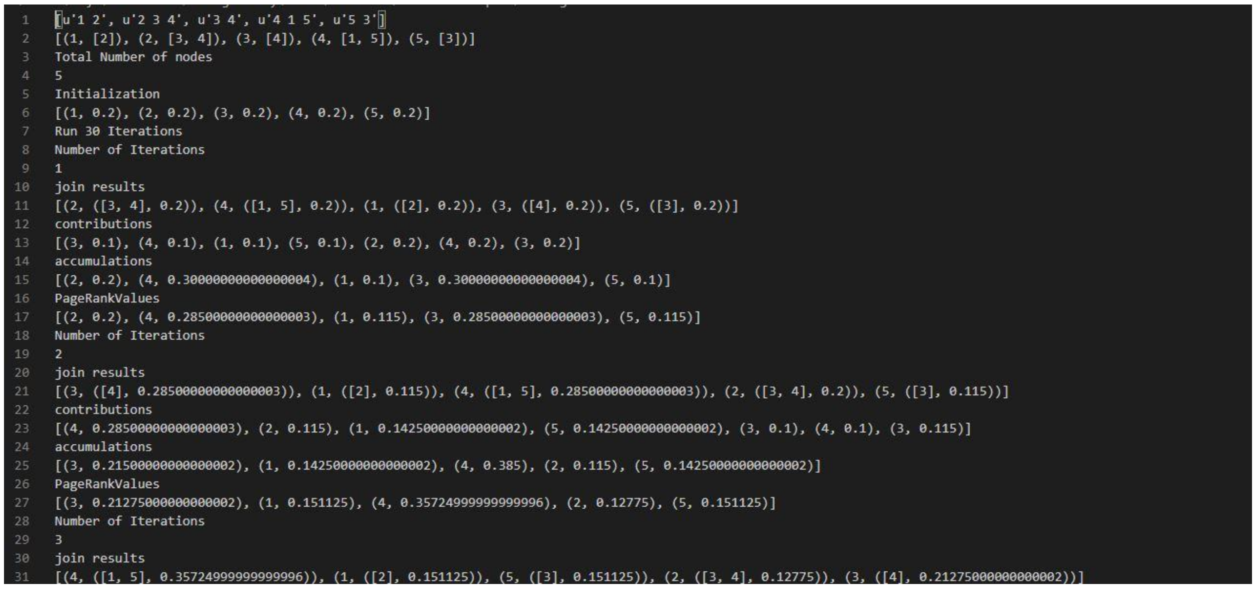
2.1) Screenshots of the key steps. For example, the screenshot for the outputs in the terminal when you run the command. It will demonstrate that your program has no bug.

Command to run the .py file



```
PS C:\Users\Joju\Documents\College-Danny\2020\BIG DATA\Homework 5\Homework 5> spark-submit PageRank.py > PageRank.txt
```

Output: PageRank.txt file



```
[u'1 2', u'2 3 4', u'3 4', u'4 1 5', u'5 3']
[(1, [2]), (2, [3, 4]), (3, [4]), (4, [1, 5]), (5, [3])]
Total Number of nodes
5
Initialization
[(1, 0.2), (2, 0.2), (3, 0.2), (4, 0.2), (5, 0.2)]
Run 30 Iterations
Number of Iterations
1
join results
[(2, ([3, 4], 0.2)), (4, ([1, 5], 0.2)), (1, ([2], 0.2)), (3, ([4], 0.2)), (5, ([3], 0.2))]
contributions
[(3, 0.1), (4, 0.1), (1, 0.1), (5, 0.1), (2, 0.2), (4, 0.2), (3, 0.2)]
accumulations
[(2, 0.2), (4, 0.30000000000000004), (1, 0.1), (3, 0.30000000000000004), (5, 0.1)]
PageRankValues
[(2, 0.2), (4, 0.28500000000000003), (1, 0.115), (3, 0.28500000000000003), (5, 0.115)]
Number of Iterations
2
join results
[(3, ([4], 0.28500000000000003)), (1, ([2], 0.115)), (4, ([1, 5], 0.28500000000000003)), (2, ([3, 4], 0.2)), (5, ([3], 0.115))]
contributions
[(4, 0.28500000000000003), (2, 0.115), (1, 0.14250000000000002), (5, 0.14250000000000002), (3, 0.1), (4, 0.1), (3, 0.115)]
accumulations
[(3, 0.21500000000000002), (1, 0.14250000000000002), (4, 0.385), (2, 0.115), (5, 0.14250000000000002)]
PageRankValues
[(3, 0.21275000000000002), (1, 0.151125), (4, 0.35724999999999996), (2, 0.12775), (5, 0.151125)]
Number of Iterations
3
join results
[(4, ([1, 5], 0.35724999999999996)), (1, ([2], 0.151125)), (5, ([3], 0.151125)), (2, ([3, 4], 0.12775)), (3, ([4], 0.21275000000000002))]
```

2.2) Explain your results. Does your implementation give the exact PageRank values?

Yes, it gives the exact PageRank Values. It matches the samples output file.