**Data Science (ITE4005)**

**Programming Assignment #3**

**Clustering : DBSCAN**



컴퓨터공학부 컴퓨터전공

2009004065 유건열

[1. Environment 2](#_Toc482923333)

[2. Summary of algorithm 3](#_Toc482923334)

[3. Detailed Description of codes 3](#_Toc482923335)

[(1) Main flow 3](#_Toc482923336)

[(2) Detail description 4](#_Toc482923337)

[ Imported Libraries 4](#_Toc482923338)

[ Class for Object Node 4](#_Toc482923339)

[ Handling user’s argument 4](#_Toc482923340)

[ Getting input file number 5](#_Toc482923341)

[ Opening files 5](#_Toc482923342)

[ Reading the data of points in input file 6](#_Toc482923343)

[ Connecting neighbor points 6](#_Toc482923344)

[ Setting recursion limit bigger 7](#_Toc482923345)

[ Do clustering 7](#_Toc482923346)

[ Expanding cluster 8](#_Toc482923347)

[ Sorting clusters 9](#_Toc482923348)

[ Writing clusters into output files 9](#_Toc482923349)

[ Closing files 9](#_Toc482923350)

[4. Instructions for executing the source code 9](#_Toc482923351)

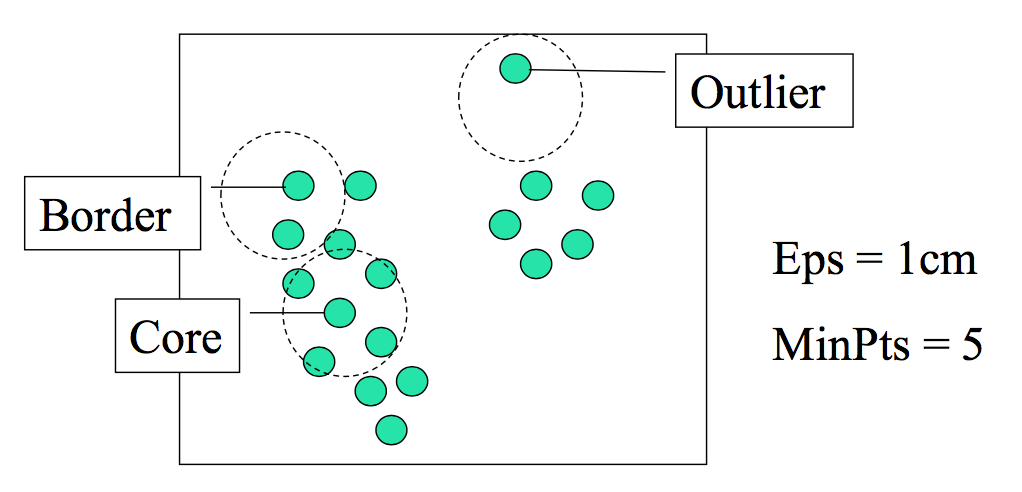
[5. Result of test 11](#_Toc482923352)

# 1. Environment

- OS : OS X

- Language : Python 2.7.12

# 2. Summary of algorithm



The objective of this assignment is to make clusters using DBSCAN. DBSCAN is density-based clustering method. To understand DBSCAN, we have to know some terms.

* Eps : maximum radius of the neighborhood

DBSCAN proceeds with ‘neighbor object’. If the distance of two points is less than eps, they are neighbors to each other.

* MinPts : minimum number of points in an Eps-neighborhood a given point

To make a cluster, the object has to be a core point. MinPts is the criteria to decide if it is a core point or not. If a point has neighbor points more than minPts, the point is the core point.

* Directly density-reachable

If point p has neighbor points more than minPts, (Of course, they are ‘neighbor points’, so their distances are less than eps.) we can say every neighbor point of point p is ‘directly density-reachable’. I think if any neighbor point is directly density-reachable from a target point, it can be expressed like ‘The target point is a core point in DBSCAN.’

So, the rule is very simple. If the point is a core point, a cluster will be formed, or expanded.

# 3. Detailed Description of codes

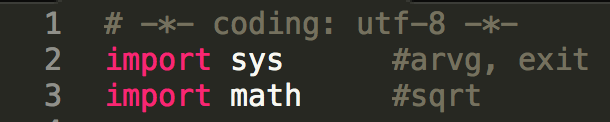
## (1) Main flow

In DBSCAN, when an object expands the cluster, we should find its neighbors. But, if the program calculates every distance between the all pairs whenever expanding the cluster, the time complexity will be awful.

So, my program connects every neighbor first. By scanning DB once, if the distance of two points is less than eps. I saved this connectivity in ‘pointing’ list. It is the attribute of ‘Object node’ instance. After connecting, the program easily decides if each point is a core point or not. If the length of ‘pointing list’ is bigger than minPts, the target point is a core point. If this point is the first point of the cluster, the point creates a cluster. Unless, the point expands a cluster.

## (2) Detail description

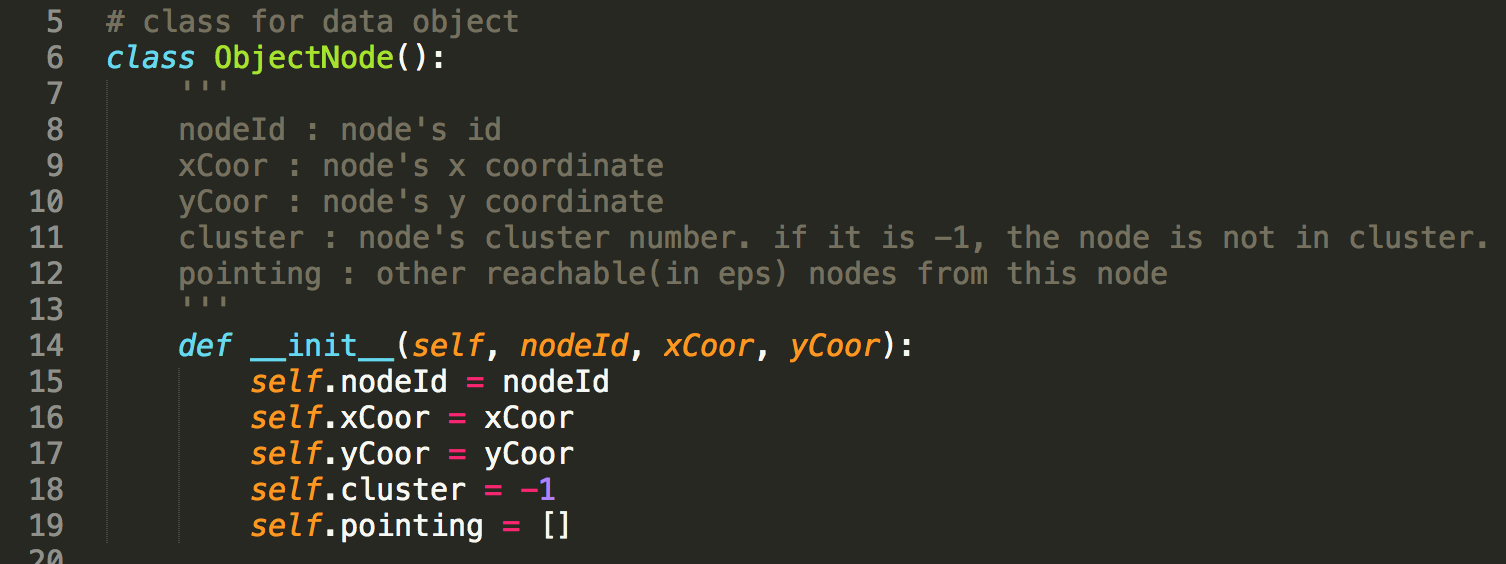
### Imported Libraries



I imported sys for ‘arvg’ that is for user’s argument, and for exit.

And I imported math for ‘sqrt’. Because I have to calculate the Euclidean distance of the pairs of points.

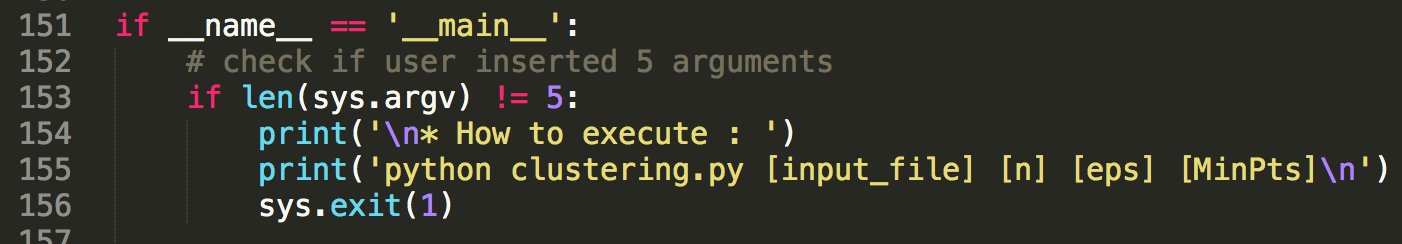
### Class for Object Node



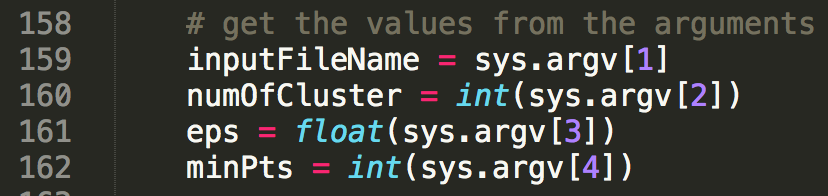
This is ‘ObjectNode’ class in my program. Every point in input data will be an instance of this ‘ObjectNode’.

* nodeId : the id of node. This is the first column of input file.
* xCoor : node’s x coordinate. This is the second column of input file.
* yCoor : node’s y coordinate. This is the third column of input file.
* cluster : node’s cluster id. Default value is -1. At the end of the process, if this value is still -1, the node is not in cluster.
* pointing : This list saves neighbor nodes of this node. If a node has neighbor points in this ‘pointing list’ more than eps, this node is a core point.

### Handling user’s argument

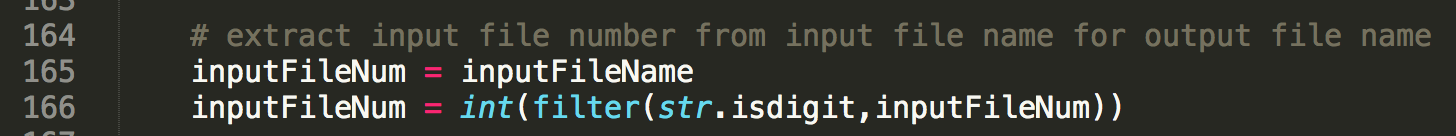


In main function, it checks the arguments’ number first. We need five argument including file name. (input file name, the number of clusters, eps, min pts). If the number of arguments is not 5, it shows ‘how to execute’ and the program will be exit.



If user inserts five arguments exactly, we can get input file name, the number of clusters, eps, minPts.

### Getting input file number



The output file names are like this. :

input#\_cluster\_0.txt,

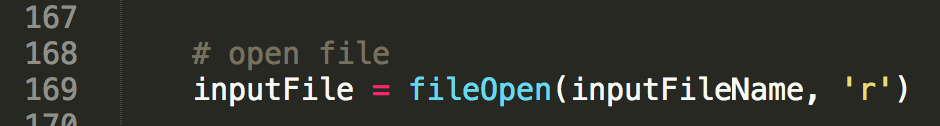
input#\_cluster\_1.txt,

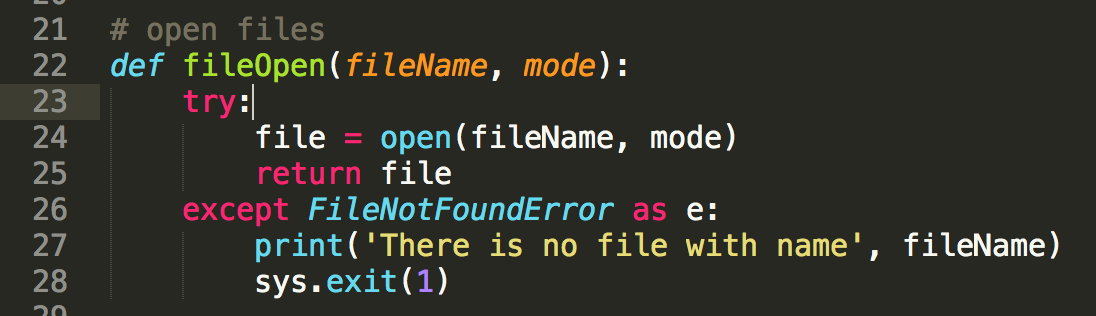
…

input#\_cluster\_n-1.txt

# is the input file number. So, for the output files, I had to get the input file number. Input file name format is like this. : ‘input#.txt’, where # is number. Therefore, I need to extract the number from the input file number. ‘filter’ function can do this.

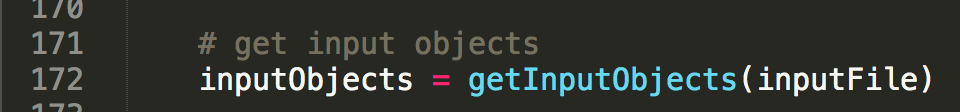
### Opening files

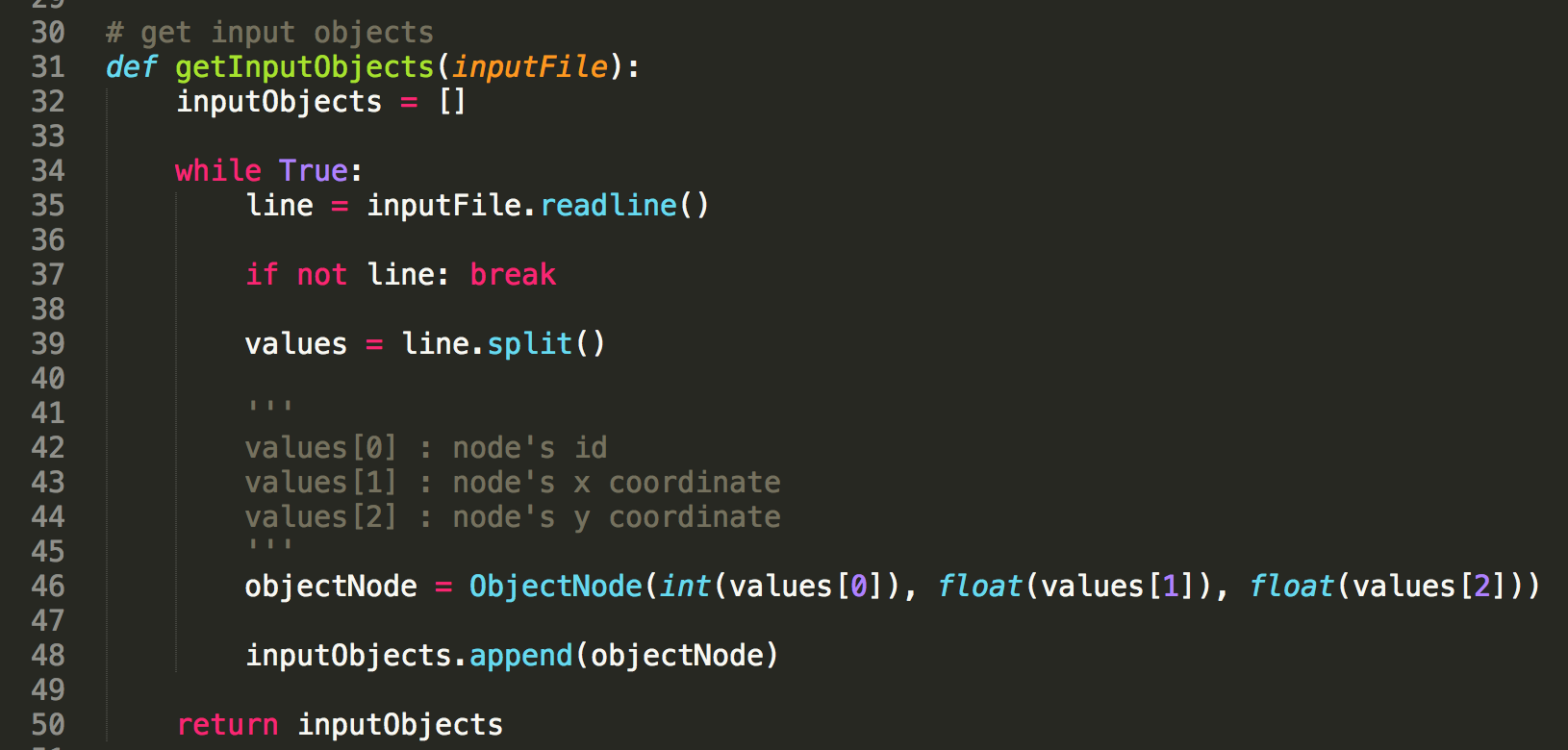




And, it opens files for input data, and output data. ‘fileOpen’ function will do that. If ‘FileNotFoundError’ occurs, program will be exit. (If user insert file name is not in directory.)

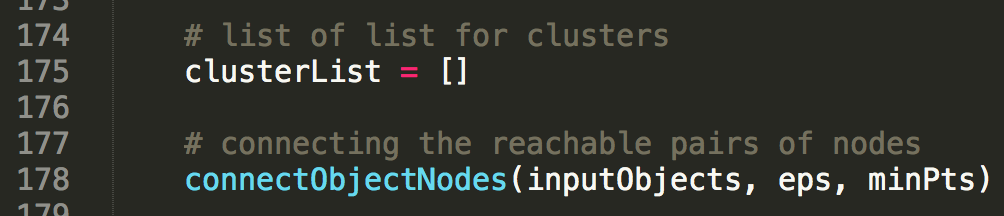
### Reading the data of points in input file

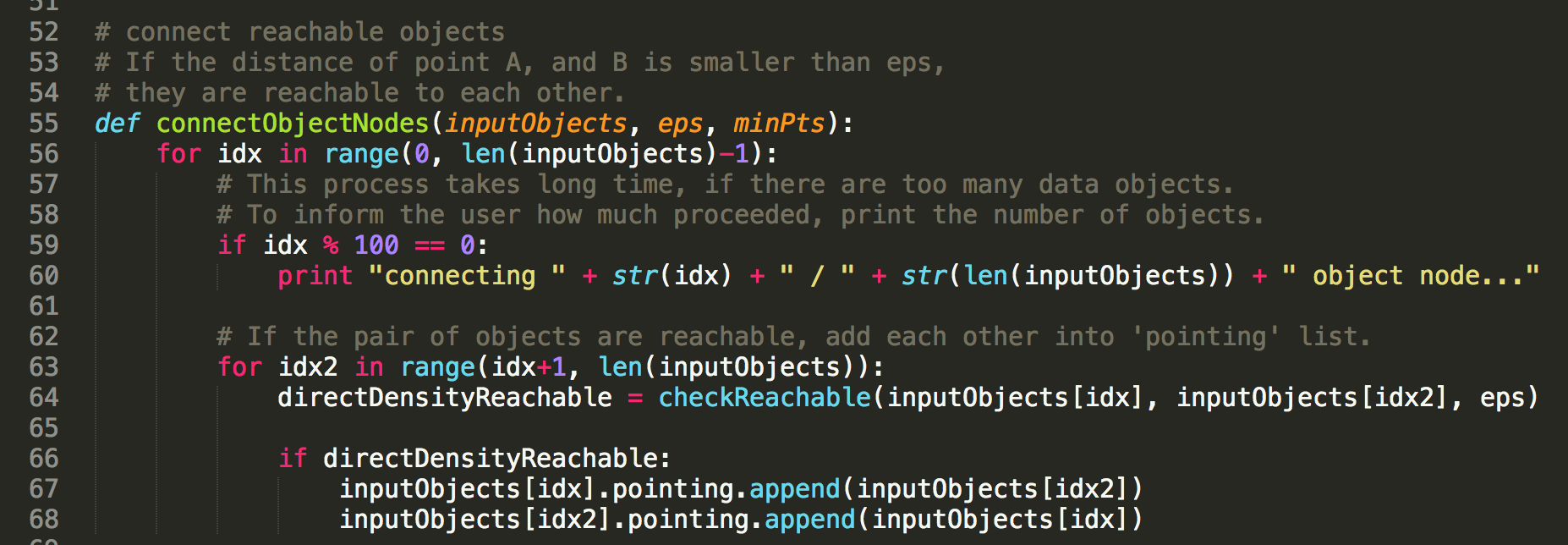




‘getInputObjects’ method reads every line from input file. Each line has node’s id, x coordinate, and y coordinate. ‘inputObjects’ list saves every ‘ObjectNode’ instance. These instances generate with above separated values.

### Connecting neighbor points



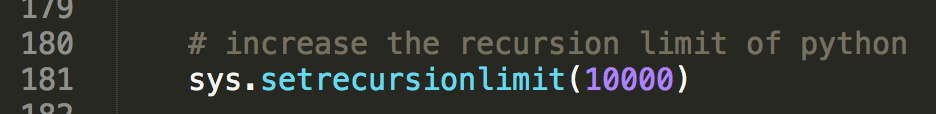


Before connecting neighbor points, it makes the list for cluster.

‘connectObjectNodes’ method connects every pair of neighbors. This method works like bubble sort. Outer for statement’s range is 0~n-1. And inner for statement’s range is outer idx+1 ~ n. Because ‘neighbority’ is symmetric attribute. For example, if A object is neighbor of B, B is neighbor of A, as well. So, we don’t need to check it again for reverse case. This process takes long time. So, I printed the object number in every 100th point.

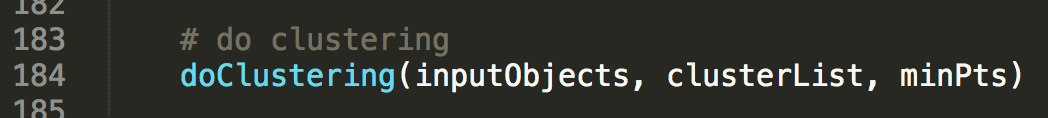
‘checkReachable’ function decides if the pairs are neighbors or not. If they are neighbors, they save each other in pointing list.

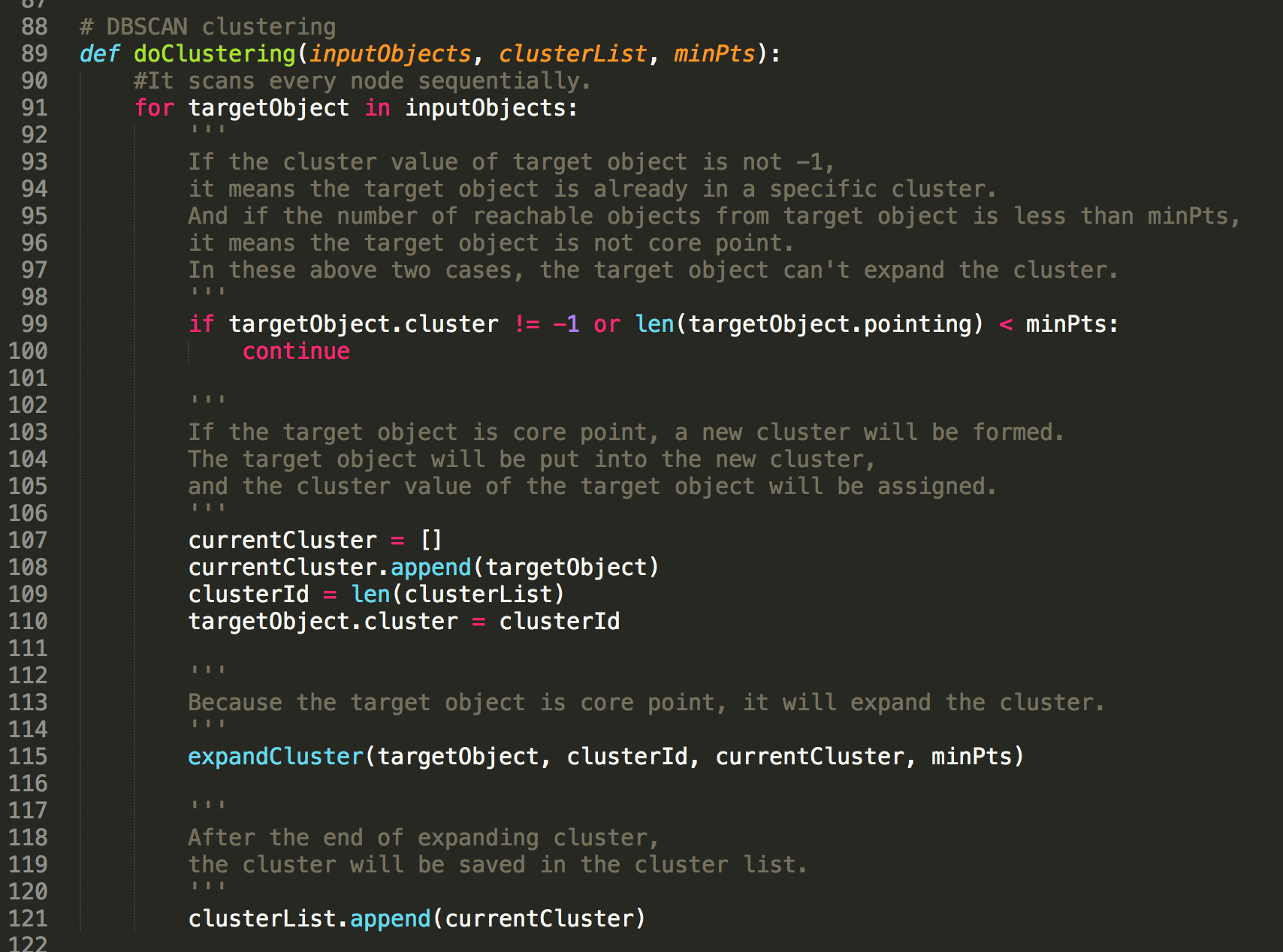
### Setting recursion limit bigger



Python has about 1000 for the limitation for recursion. But I use recursion in this program. And this program needs bigger recursion limitation. So, I used ‘setrecursionlimit’ method for fixing this problem.

### Do clustering



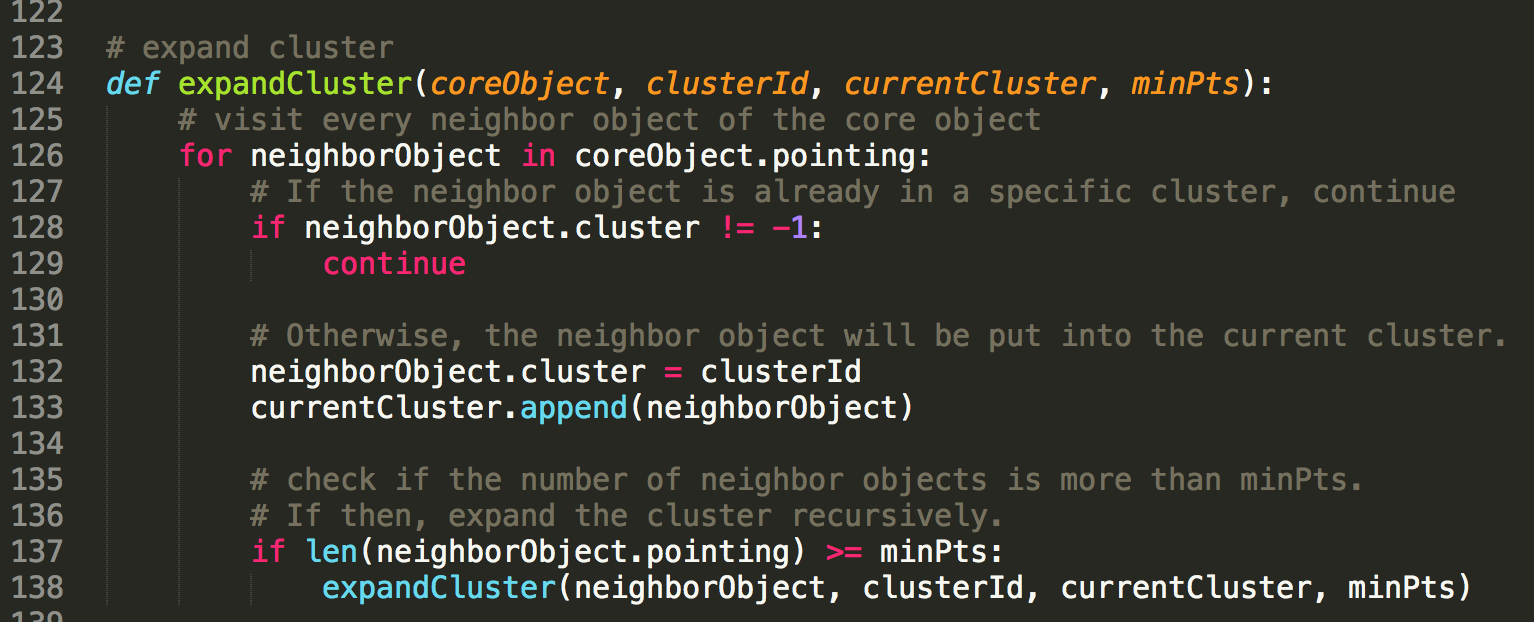


‘doClustering’ method scans every point sequentially. First, check the point’s cluster value. If the cluster value is not -1, it means the target point is already in a specific cluster. So, we don’t need to care about this point any more. And the number of neighbor points is less than minPts, the target point is not a core point. Therefore, the point can’t create a cluster.

If the target point is a core point, a new cluster will be formed. ‘currentCluster’ list is initialized here. This will be used to save points contained in current cluster. And the target point is put in this cluster. The cluster value will be assigned as well. Because we have to check the point is in a specific cluster or not.

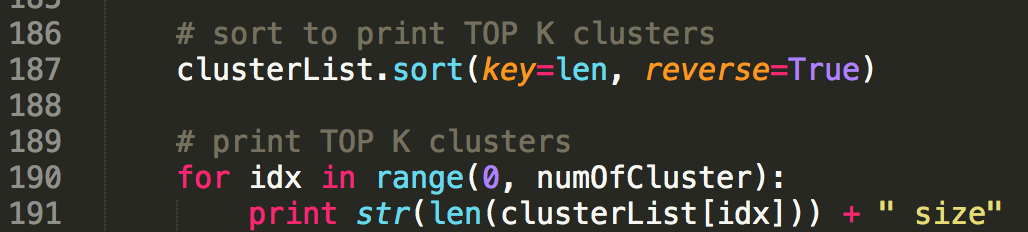
Cluster id is len(clusterList). If clusterList is empty, len(clusterList) is 0, and the new cluster id will be 0. After finding a core point, and creating a new cluster, this core point will expand this cluster. At the end of the expansion of this cluster, it saves this cluster in clusterList.

### Expanding cluster



Whenever we find a core point after creating a new cluster, the core point expands the cluster. ‘expandCluster’ method visits every neighbor point of the core point. First, it put neighbor point in cluster. Of course, if the neighbor point is already in the other cluster, it skips it. After putting in cluster, check whether the neighbor point is a core point or not by the number of neighbor points of this neighbor point. If the point is a core point, it expands the cluster recursively.

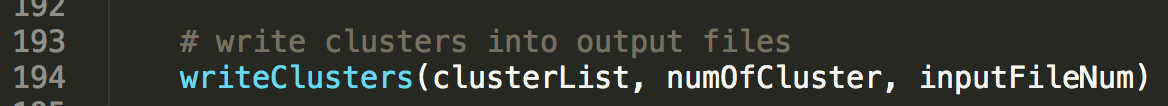
### Sorting clusters

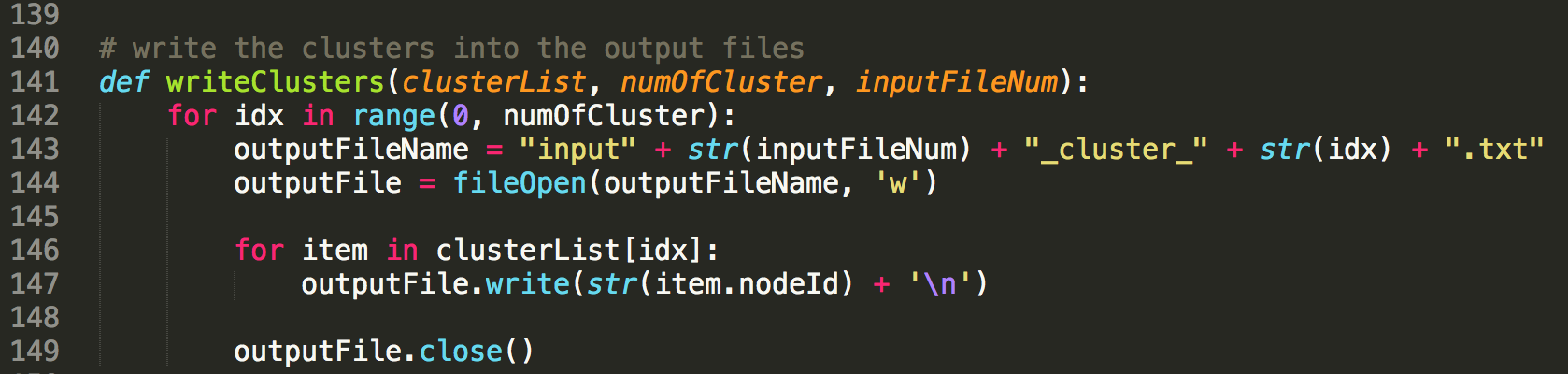


In some cases, there can be some outliers, or small clusters. I wanted to get the TOP K(number of clusters) biggest clusters. So, it sorts the clusters by its length. ‘reverse=True’ statement makes this cluster list descending order.

I printed the clusters’ size to check if the program works well.

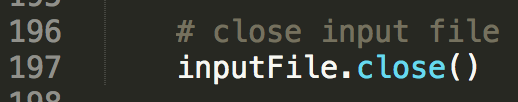
### Writing clusters into output files





Output files will be generated in the same number with the clusters’ number. The output file name contains the input file number and cluster number. After opening the output file, write every node id in that cluster. Then, close that file.

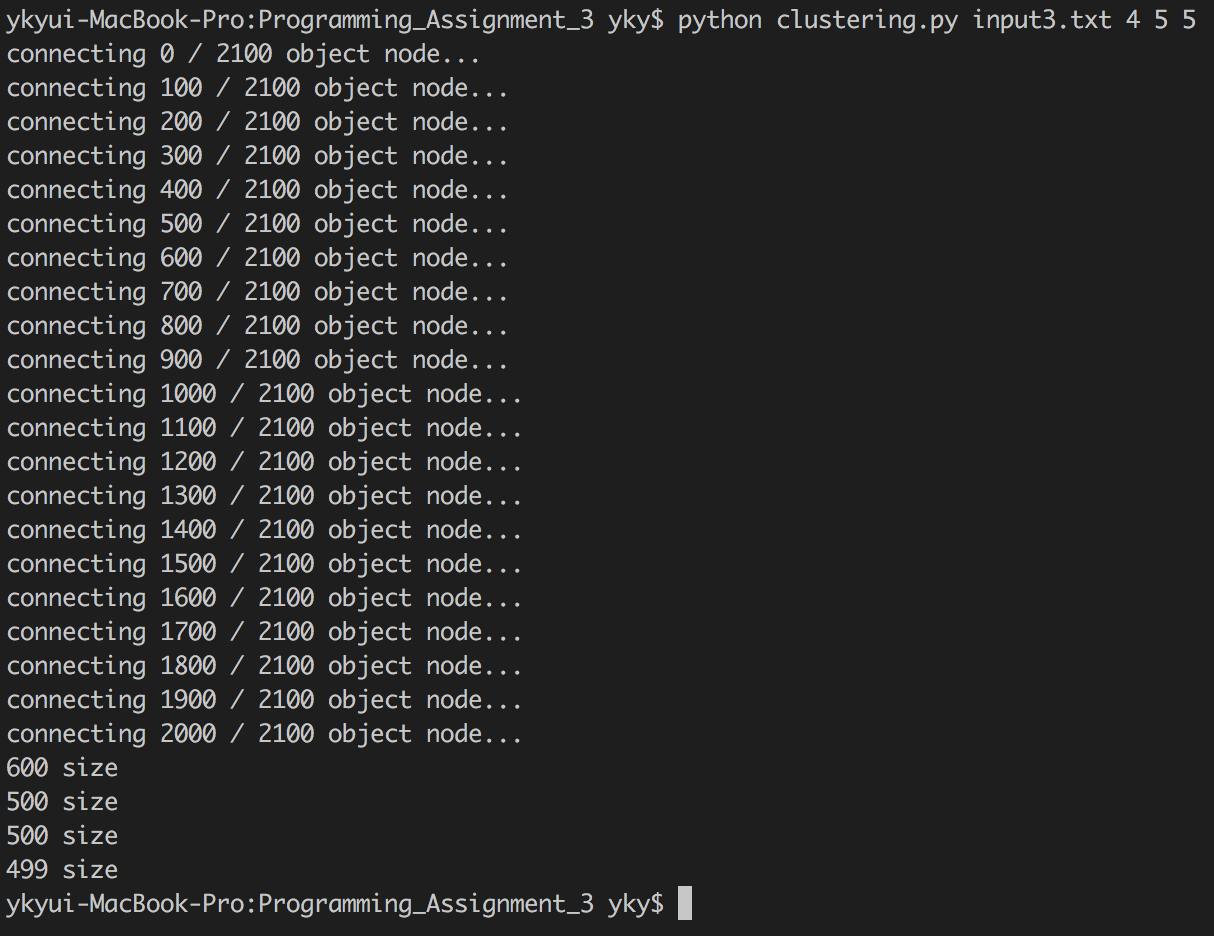
### Closing files



At the end of the program, it closes the input file.

# 4. Instructions for executing the source code

I used OS X, and python. So, you don’t need to compile it into exe file. You can execute it just by python command.



Just input this command in the directory that has clustering.py, and input files.

$ python clustering.py [input\_file\_name] [number of clusters] [eps] [minPts]

# 5. Result of test

