COMP1811 – Python Project Report

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** |  | **Student ID** |  |

1. BRIEF STATEMENT OF FEATURES YOU HAVE COMPLETED

|  |  |  |  |
| --- | --- | --- | --- |
| 1.1 Circle the parts of the coursework you have **fully completed and are fully working**. Please be accurate. | **Features F1:** i ☐  **F2:** i ☐  **F3:** i ☐ | ii ☐ iii ☐  ii ☐  ii ☐ | iv ☐ |
| 1.2 Circle the parts of the coursework you have | **Features F1:** i ☐  **F2:** i ☐ | ii ☐ iii ☐  ii ☐ | iv ☐ |
| **partly completed or are partly working.** |
|  | **F3:** i ☐ | ii ☐ |  |
| Briefly explain your answer if you circled any parts in 1.2 | | | |

1. CONCISE LIST OF BUGS AND WEAKNESSES

*A concise list of bugs and/or weaknesses in your work (if you don't think there are any, then say so). Bugs that are declared in this list will lose you fewer marks than ones that you don't declare! (****100-200 word****, but word count depends heavily on the number of bugs and weaknesses identified.)*

* 1. BUGS

*I don’t think there are any bugs in my code*

# WEAKNESSES

* *getGroups(self) function in the network class is not a good function because it is returning two things one the group list and other is the number of polygons in the network. A good function return only one thing.*
* *Variables are not private. Privatization allows you to hide the internal workings of a class, exposing only the necessary and relevant functionalities through public interfaces. This reduces complexity and enhances code readability by providing a clear separation between the interface and the implementation.*
* *It consider input as case sensitive. For example John and john are different for this code.*
* *In the shape feature as we have seen that pripod and triangle are the same so I only considered triangle and did not print the tripod shape.*

1. DESCRIPTION OF THE FEATURES IMPLEMENTED

# Friend Group Size Calculation:

*The getGroups method is implemented to calculate the sizes of friend groups in the network. It utilizes a breadth-first search (BFS) approach to traverse the network and count the size of each group. The method initializes a visited dictionary to keep track of visited members and a groups\_sizes list to store the count of friend groups of different sizes.*

*Within the BFS loop, it checks if the current member has been visited. If not, it starts a new BFS traversal from that member. While traversing, it increments the gz variable to count the size of the group. If the size is greater than or equal to 5, it increments the count of groups with 5 or more members (groups\_sizes[4]), otherwise, it increments the count for the respective group size.*

*At the end of the loop, the method returns the groups\_sizes list, representing the count of friend groups of different sizes.*

# Polygon Calculation:

*The getGroups method also counts the number of polygons in the network. It introduces a polygons variable initialized to 0. During the BFS traversal, if a group has exactly 4 members and all of them have exactly 3 friends, it increments the polygons count. The method then returns both the groups\_sizes list and the polygons count.*

# Star Calculation:

*The getStars method is implemented to count the number of star-shaped groups in the network. It initializes a stars variable to 0 and iterates through each member. If a member has exactly 4 friends and all of their friends have only 1 friend (except the current member), it increments the stars count. Finally, the method returns the count of star-shaped groups.*

# Printing Group Sizes:

*The printGroupSize method calls the getGroups method to obtain the count of friend groups of different sizes. It then iterates over the groups\_sizes list and prints the number of groups with each specific size.*

# Printing Network Shape:

*The printShape method also utilizes the getGroups method to obtain the count of friend groups of different sizes and the number of polygons. It also calls the getStars method to count the number of star-shaped groups. It then prints the counts of singleton, pair, triangle, polygon, and star shapes.*

*These features are implemented well and integrated into the Network class. They provide valuable insights into the network structure, including the sizes of friend groups, the presence of polygons, and the count of star-shaped groups.*

1. CLASSES AND OOP FEATURES

*List all the classes used in your program and include the attributes and behaviours for each. You may use a class diagram to illustrate these classes. Your narrative for section 3.2 should describe the design decisions you made and the OOP techniques used. List the classes you developed and provide an exposition on the choice of classes, class design and OOP features implemented. (****200-400 words****).*

# CLASSES USED

NetworkObject (Abstract Class):

Attributes: None

Behaviors:

\_\_str\_\_(): An abstract method to provide a string representation of the object.

size(): An abstract method to calculate the size or length of the object.

Member (Derived from NetworkObject):

Attributes:

name: A string representing the name of the member.

friendList: A list storing the friends of the member.

Behaviors:

\_\_init\_\_(self, name): Initializes a Member object with a given name and an empty friend list.

getFriends(self): Returns the friend list of the member.

addFriend(self, friend): Adds a friend to the friend list.

\_\_str\_\_(self): Provides a string representation of the member and their friends.

deleteFriend(self, friend): Removes a friend from the friend list.

size(self): Calculates the size of the friend list.

Network (Derived from NetworkObject):

Attributes:

network: A dictionary storing member names as keys and corresponding Member objects as values.

Behaviors:

\_\_init\_\_(self): Initializes a Network object with an empty network dictionary.

addMember(self, member): Adds a member to the network.

addFriend(self, member, friend): Adds a friend to a member's friend list.

size(self): Calculates the size of the network.

deleteMember(self, member): Deletes a member from the network and removes them from their friends' friend lists.

getGroups(self): Calculates the sizes of friend groups and the number of polygons in the network.

printGroupSize(self): Prints the sizes of friend groups in the network.

printShape(self): Prints the shapes (singleton, pair, triangle, polygon, star) in the network.

getStars(self): Calculates the count of star-shaped groups in the network.

\_\_str\_\_(self): Provides a string representation of the network.

* 1. BRIEF EXPLANATION OF CLASS DESIGN AND OOP FEATURES USED

The choice of classes in the implementation follows an object-oriented approach. The NetworkObject class serves as an abstract base class to provide common attributes and behaviors for the derived classes (Member and Network). By inheriting from NetworkObject, the derived classes can implement their specific functionality while adhering to the same interface.

The Member class encapsulates the data related to an individual member, such as their name and friend list. It provides methods to add and delete friends, retrieve the friend list, calculate the size, and generate a string representation. This class follows the principles of encapsulation and information hiding by keeping the friend list as a private attribute and exposing necessary functionality through public methods.

The Network class represents the overall network of members. It uses a dictionary to store Member objects, enabling efficient member lookup and retrieval. The class provides methods to add members, establish friendships, delete members, calculate network statistics, and generate a string representation. The use of a dictionary enhances the efficiency of member management in the network.

1. CODE FOR THE CLASSES CREATED

*Add the* ***code for each of the classes you have implemented yourself*** *here. If you have contributed to parts of classes, please highlight those parts in a different colour. Copy and paste relevant code - actual code please, no screenshots! Make it easy for the tutor to read. Add explanation if necessary – though your in-code comments should be clear enough. You will lose marks if screenshots are provided instead of code.*

*(DO NOT provide a listing of the entire code. You will be marked down if a full code listing is provided.)*

* 1. NetworkObject

class NetworkObject(ABC):

    @abstractmethod

    def \_\_str\_\_(self):

        pass

    @abstractmethod

    def size(self):

        pass

* 1. Member(NetworkObject)

class Member(NetworkObject):

    def \_\_init\_\_(self, name):

        super().\_\_init\_\_()

        self.name = name

        self.friendList = []

    def getFriends(self):

        return self.friendList

    def addFriend(self, friend):

        self.friendList.append(friend)

    def \_\_str\_\_(self):

        r = f"{self.name} is friends with "

        if self.size() == 0:

            return r + "no one"

        for friend in self.friendList:

            r += f"{friend}, "

        return r[:-2]

    def deleteFriend(self, friend):

        self.friendList.remove(friend)

def size(self):

        return len(self.friendList)

* 1. Network Class

class Network(NetworkObject):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.network = {}

    def addMember(self, member):

        self.network[member] = Member(member)

    def addFriend(self, member, friend):

        self.network[member].addFriend(friend)

    def size(self):

        return len(self.network)

    def deleteMember(self, member):

        for friend in self.network[member].getFriends():

            self.network[friend].deleteFriend(member)

        del(self.network[member])

    def getGroups(self):

        groups\_sizes = [0]\*5

        visited = defaultdict(lambda: False)

        polygons = 0

        for member in self.network:

            if visited[member] == False:

                queue = [member]

                gz = 0

                is\_polygon = True

                while len(queue) != 0:

                    cur = queue.pop(0)

                    if visited[cur] == True:

                        continue

                    visited[cur] = True

                    neighbors = self.network[cur].getFriends()

                    if self.network[cur].size() != 3:

                        is\_polygon = False

                    for neighbor in neighbors:

                        queue.append(neighbor)

                    gz += 1

                # print(gz)

                if gz >= 5:

                    groups\_sizes[4] += 1

                else:

                    groups\_sizes[gz-1] += 1

                if is\_polygon and gz == 4:

                    polygons += 1

        return groups\_sizes, polygons

    def printGroupSize(self):

        groups\_sizes, \_ = self.getGroups()

        print("The size of  friend groups:")

        for i, size in enumerate(groups\_sizes):

            print(f"Number of  groups with {i+1} members: {size}")

    def printShape(self):

        groups\_sizes, polygons = self.getGroups()

        stars = self.getStars()

        print("Friend group network shapes:")

        print(f"{groups\_sizes[0]} singleton")

        print(f"{groups\_sizes[1]} pair")

        print(f"{groups\_sizes[2]} triangle")

        print(f"{polygons} polygon")

        print(f"{stars} star")

    def getStars(self):

        stars = 0

        for member in self.network:

            if self.network[member].size() == 4:

                is\_star = True

                for friend in self.network[member].getFriends():

                    if self.network[friend].size() != 1:

                        is\_star = False

                        break

                if is\_star == True:

                    stars += 1

        return stars

    def \_\_str\_\_(self):

        r = ""

        if self.size() == 0:

            return ""

        for member in self.network:

            r += self.network[member].\_\_str\_\_() + "\n"

        return r[:-1]

    def averageFriends(self):

        sumi = 0

        for member in self.network:

            sumi += self.network[member].size()

        return sumi / self.size()

    def maxFriends(self):

        maxi = 0

        name = "No One"

        for member in self.network:

            if self.network[member].size() >= maxi:

                maxi = self.network[member].size()

                name = member

        return name

* 1. MakeNetwork

class MakeNetwork:

    def \_\_init\_\_(self, filename):

        self.filename = filename

    def make(self):

        file = open(self.filename, "r")

        n = int(file.readline())

        line = file.readline()

        nw = Network()

        while line != "":

            line = line.rstrip().split()

            if len(line) == 1:

                nw.addMember(line[0])

            else:

                f1, f2 = line

                try:

                    nw.addFriend(f1, f2)

                except:

                    nw.addMember(f1)

                    nw.addFriend(f1, f2)

                try:

                    nw.addFriend(f2, f1)

                except:

                    nw.addMember(f2)

                    nw.addFriend(f2, f1)

            line = file.readline()

        if nw.size() != n:

            raise Exception("Member defined in the file are not equal to the actual members")

        return nw

1. TESTING

*Firstly I created 4 test files having different networks and then check every feature on these files. File names are nw1, nw2, nw3, nw4. Following are the input files and the output of the programme.*

# Nw1.txt

*10*

*Alice Bob*

*David Emily*

*Emily Frank*

*Frank George*

*George Helen*

*Helen Ian*

*Ian Jane*

*Alice David*

*Bob Emily*

*David George*

*Emily Helen*

*Frank Ian*

*George Jane*

*Ian Alice*

*Jane Bob*

*Alice Frank*

*Bob George*

*David Ian*

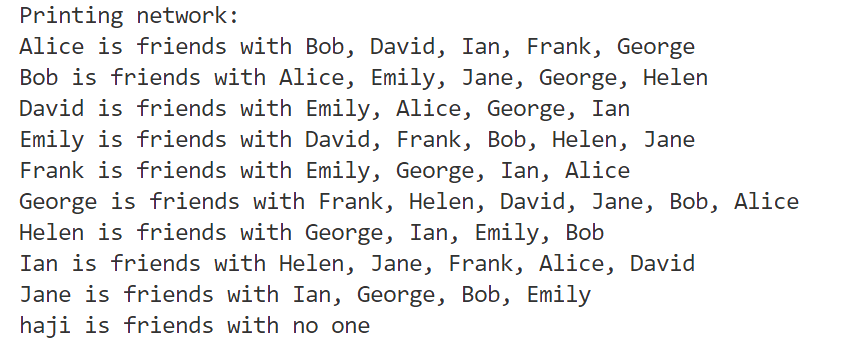
*Emily Jane*

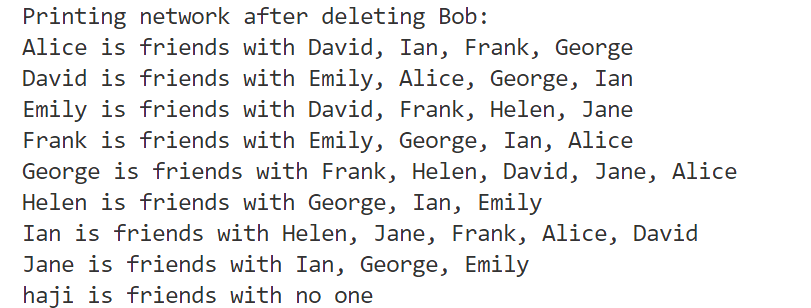
*George Alice*

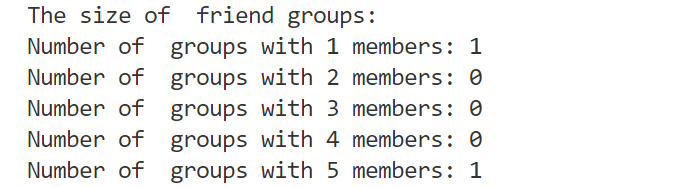
*Helen Bob*

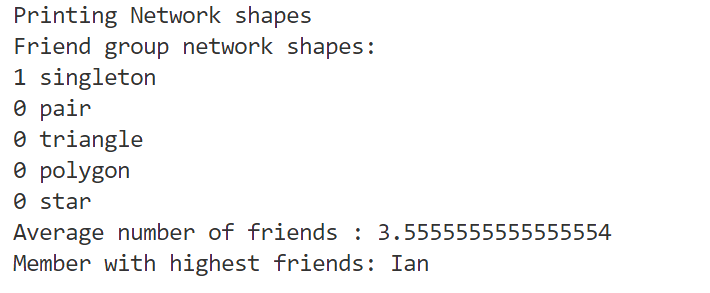
*Haji*

# Results



**

**

**

# Nw4.txt

11

adam chris

Bob amir

Bob mia

chris liz

jon chris

chris zia

mia amir

mo ala

ali

# Results:

# 

# 

# 

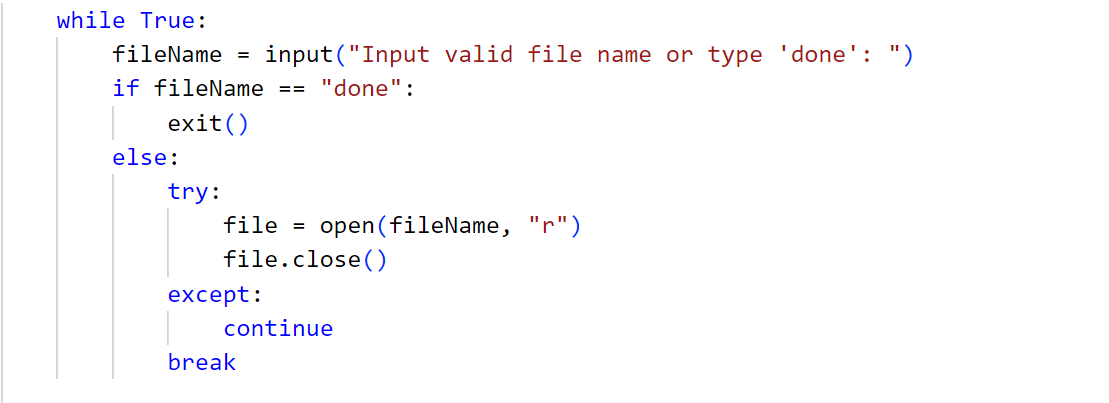
Same way there are other input files that I have created for the testing of this programme.

1. ANNOTATED SCREENSHOTS DEMONSTRATING IMPLEMENTATION

*Provide screenshots that demonstrate the features implemented. Annotate each screenshot and if necessary, provide a brief description for* ***each*** *(****up to 100 words****) to explain the code in action. Make sure the screenshots make clear what you have implemented and achieved.*

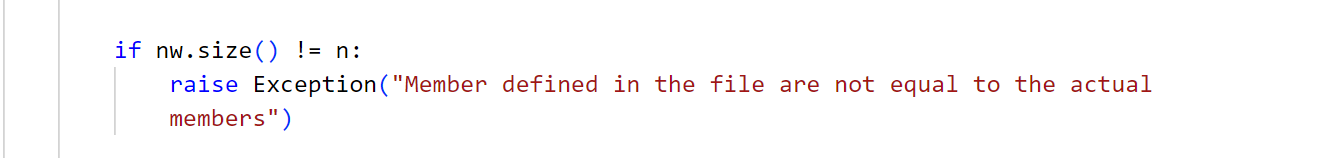
* 1. FEATURE F1

# SUB-FEATURE I- SCREENSHOTS …



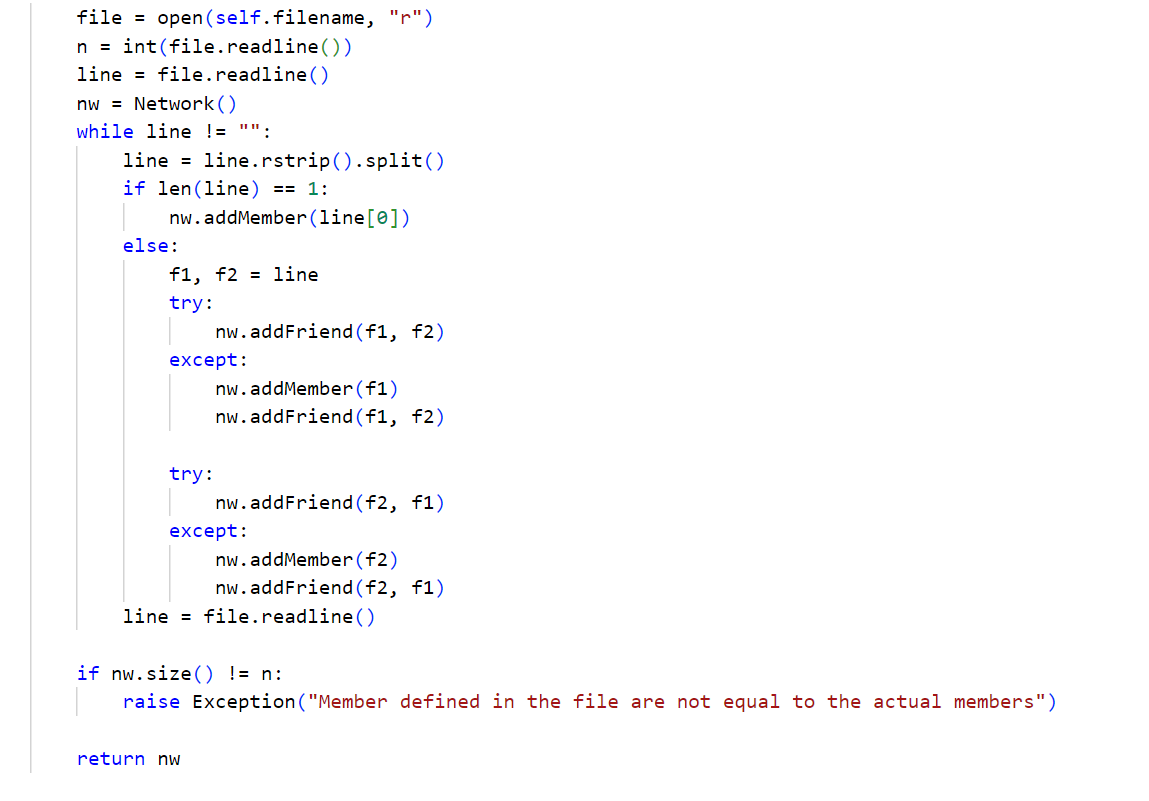
While loop make sure that the code runs while an existing file name is given to the input stream. Then it checks if the input stream is ‘done’ then it exists the programme otherwise is continue to require an input. Then it try to open a the file, if it open successfully, loop breaks.

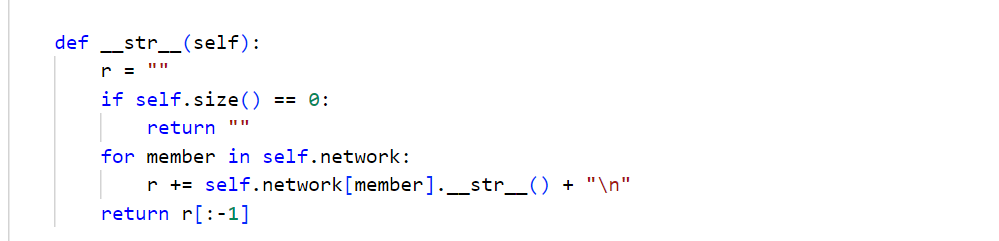
1. SUB-FEATURE II- SCREENSHOTS



Here in the make network class I check wether the number of members in the network are equal to the number of members written on top of the file.

# SUB-FEATURE III- SCREENSHOTS …

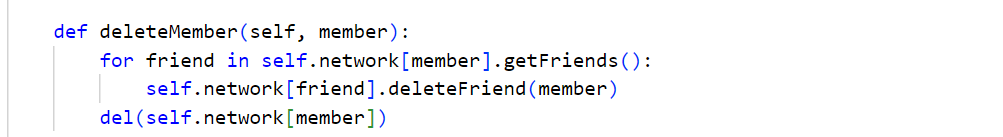




Reading the data from the file is done in the makeNetwork class. It first opens the file then read the file line by line and checks if the number of words are 2 or 1 if one then it just add a member to the network otherwise first check if the member is already in the network by adding a friend to that member and add both of these in each other friend list. At last is returned as a network object.

For printing I traversed the network and just used the \_\_str\_\_ representation of a member class object.

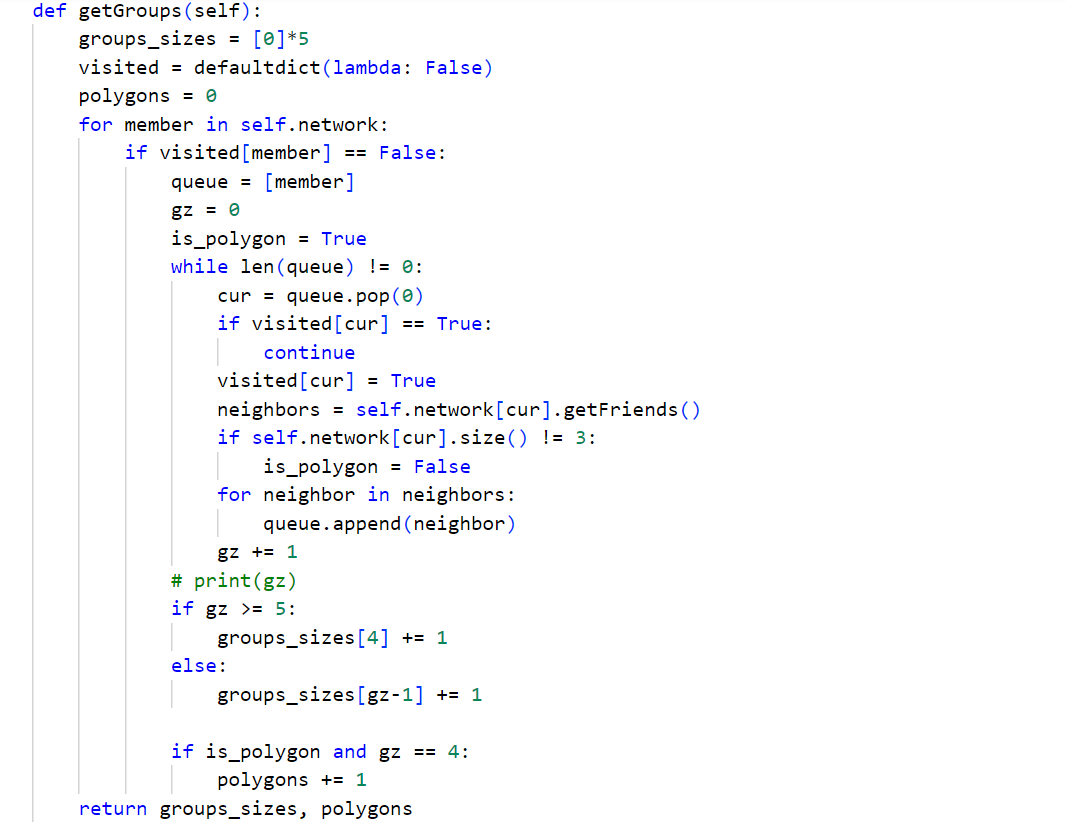
1. SUB-FEATURE IV SCREENSHOTS …

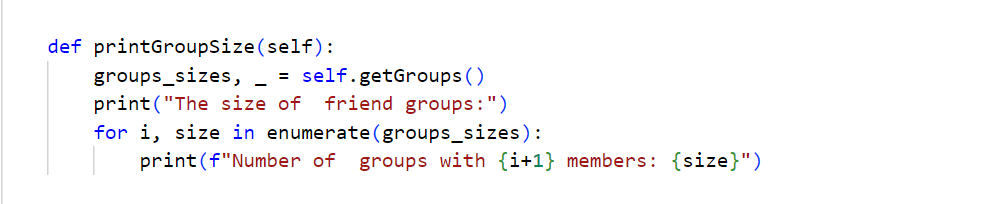


It first remove the member from its friends friend list then delete the member completely from the dictionary object. There is another deleteFriend function in the Member class ehich serves the purpose for deleting friend from the list of friends.

* 1. FEATURE F2

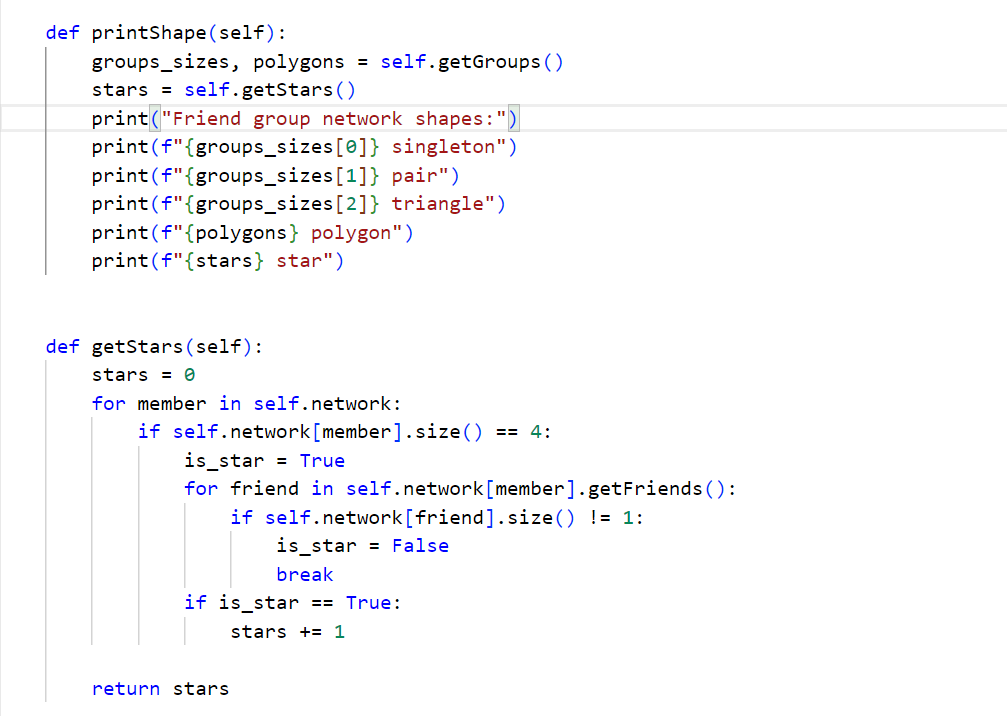
# SUB-FEATURE I- SCREENSHOTS …





This feature uses two function first is the number of groups in the network. It just traversals the network in breadth first search way. First I make a queue then add the first member in the queue then check if the cur member is visited or not. If the cur member is not visited already then it adds its neigbors to the queue and also increase the size of the group. It serves another purpose it calculate the number of the polygons groups in the network because it will be used in the next feature. Then it simply returns the list of group sizes. Then in other function I traverse on this list and print the number of groups with desired sizes.

1. SUB-FEATURE II- SCREENSHOTS …

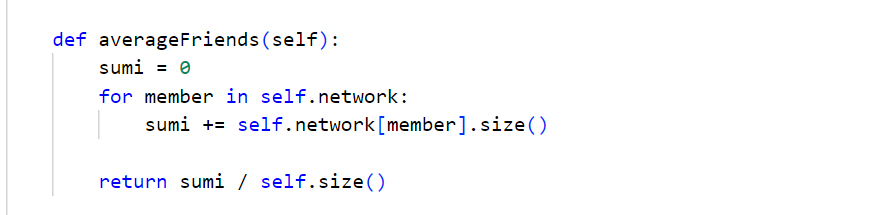


This feature uses 3 function first is the group sizes function from the previous feature and other is the get Star function. getStars() function just traverse the network and find out a member who have 4 friends and each friend only have one friend, obviously one friend means the current member in the iteration of the loop.

Then printStars() function just print the number of the shapes by getting group sizes and number of polygons from getGroupSize function and stars from getStars function.

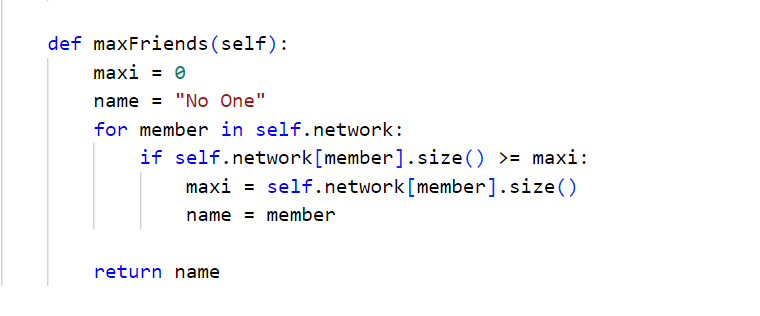
* 1. FEATURE F3

# SUB-FEATURE I- SCREENSHOTS …



This function traverse the network and just sums out the number of friends on each iteration. The n at the last it returns the sum by dividing it to the size of the network meaning the number of members in the network to get the average.

1. SUB-FEATURE II- SCREENSHOTS …



This function initialize the maxi to 0 and name of that member to no one as it does not exist. It traverse the whole network and if the number of friends on each iteration are greater than the previous maximum then it replaces the maxi with that and name of the member with previous member.

1. EVALUATION

*Give a reflective, critical self-evaluation of your experience developing the project and discuss what you would do if you had more time to work on the project. Answer the following questions for the reflection and write* ***350-400 words overall****. Please include an actual word count for this section.*

# EVALUATE HOW WELL YOUR DESIGN AND IMPLEMENTATION MEET THE REQUIREMENTS

* 1. EVALUATE YOU OWN PERFORMANCE

# WHAT WENT WELL?

Writing the code once I had made the vague picture of it in my mind. It was the easiest part of the assignment.

* + 1. WHAT WENT LESS WELL?

Writing of the input files because it required making you own network by writing line by line and it took a lot of time.

# WHAT WAS LEARNT?

I learned the making a OOP design that how to divide problem in parts then implement it using the classical object oriented structure.

* + 1. HOW WOULD A SIMILAR TASK BE COMPLETED DIFFERENTLY?

This could be completed without the usage of the classed and the objects. But it would have been very difficult.

# HOW COULD THE MODULE BE IMPROVED?

This module can be improved by making a different file for each class and then import it for the usage in the main file.

* 1. SELF-ASSESSMENT

*Please assess yourself objectively for each section shown below and then enter the total mark you expect to get. Marks for each assessment criteria are indicated between parentheses.*

CODE DEVELOPMENT (70)

# Features Implemented [40]

Feature 1 (up to 16)

Sub-features have not been implemented – 0 Attempted, not complete or very buggy – 1 to 5

Implemented and functioning without errors but not integrated – 6 to 9 Implemented and fully integrated but buggy – 10 to 14

Implemented, fully integrated and functioning without errors – 15 to 16 Feature 2 (up to 16)

Sub-features have not been implemented – 0

Attempted, not complete or very buggy – 1 to 5

Implemented and functioning without errors but not integrated – 6 to 9 Implemented and fully integrated but buggy – 10 to 14

Implemented, fully integrated and functioning without errors – 15 to 16 Feature 3 (up to 8)

Sub-features have not been implemented – 0

Attempted, not complete or very buggy – 1 to 3

Implemented and functioning without errors but not integrated – 4 to 5 Implemented and fully integrated but buggy – 6 to 7

Implemented, fully integrated and functioning without errors – 8

**For this criterion I think I got: out of 40**

# Use of OOP techniques [20]

Abstraction (up to 8)

No classes have been created – 0

Classes have been created superficially and not instantiated or used – 1 or 2

Classes have been created but only some have been instantiated and used – 3 or 4 Useful classes and objects have been created and used correctly – 5 or 6

The use of classes and objects exceeds the specification – 7 or 8 Encapsulation (up to 8)

No encapsulation has been used – 0

Class variables and methods have been encapsulated superficially – 1 to 3 Class variables and methods have been encapsulated correctly – 4 to 6 The use of encapsulation exceeds the specification – 7 to 8

Inheritance (up to 4)

No inheritance has been used – 0

Classes have been inherited superficially – 1 or 2 Classes have been inherited correctly – 3

The use of inheritance exceeds the specification – 4

Bonus marks will be awarded for the appropriate use of polymorphism (bonus marks up to 5)

**For this criterion I think I got: out of 20**

# Quality of Code [10]

Code Duplication (up to 4)

Code contains too many unnecessary code repetition – 0 Regular occurrences of duplicate code – 1

Occasional duplicate code – 2 Very little duplicate code – 3 No duplicate code – 4

PEP8 Conventions and naming of variables, methods and classes (up to 3) PEP8 and naming convention has not been used – 0

PEP8 and naming convention has been used occasionally – 1 PEP8 and naming convention has been used, but not regularly – 2

PEP8 convention used professionally and all items have been named correctly – 4 In-code Comments (up to 3)

No in-code comments – 0

Code contains occasional in-code comments – 1

Code contains useful and regular in-code comments – 2

Thoroughly commented, good use of docstrings, and header comments describing.py files – 3

## For this criterion I think I got: out of 10

DOCUMENTATION (20)

Design (up to 10) clear exposition about the design and decisions for OOP use The documentation cannot be understood on first reading or is mostly incomplete – 0 The documentation is readable, but a section(s) are missing – 1 to 3

The documentation is complete – 4 to 6

The documentation is complete and of a high standard – 7 to 10

# Testing (5)

Testing has not been demonstrated in the documentation – 0 Little white box testing has been documented – 1 or 2

White box testing has been documented for all the coursework – 3 or 4 White box testing has been documented for the whole system – 5

# Evaluation (5)

No evaluation was shown in the documentation – 0 The evaluation shows a lack of thought – 1 or 2 The evaluation shows thought – 3 or 4

The evaluation shows clear introspection, demonstrates increased awareness – 5

## For this criterion I think I got: out of 20

ACCEPTANCE TESTS - SCREENCAST (10)

# Screencast (up to 10)

Not submitted or no work demonstrated – 0

Work demonstrated was not up to the standard expected – 1 to 3 Work demonstrated was up to the standard expected – 4 to 7 Work demonstrated exceeded the standard expected – 8 to 10

## For this criterion I think I got: out of 10 I think my overall mark would be: out of 100

**APPENDIX A:** CODE LISTING

*Provide a complete listing of all the \*.py files in your PyCharm project. Make sure your code is well commented and applies professional Python convention (refer to* [*PEP 8*](https://www.python.org/dev/peps/pep-0008/) *for details). The code listed here must match that uploaded to Moodle. Please copy and paste the actual code – no screenshots please! You will lose marks if screenshots are provided instead of code.*