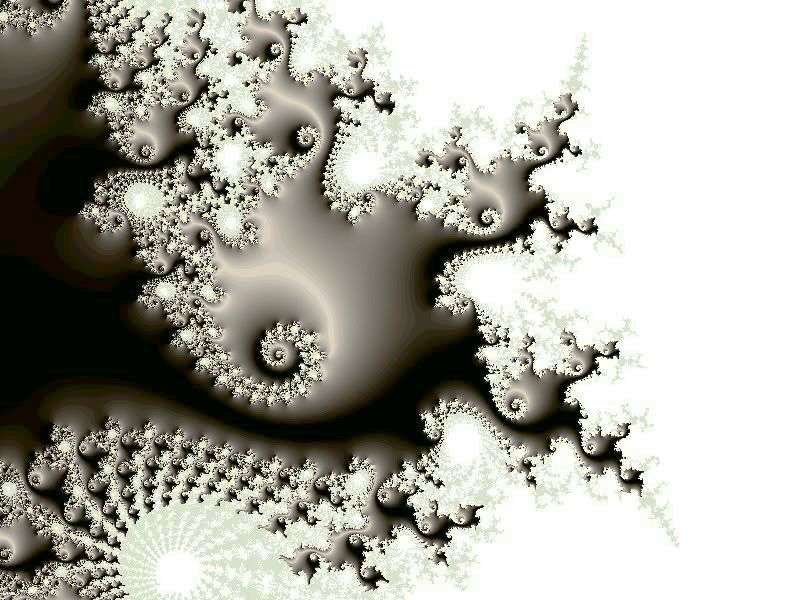
Computer Science Project



Contents

[Analysis 3](#_Toc480535417)

[Introduction 3](#_Toc480535418)

[Current System 3](#_Toc480535419)

[End User 3](#_Toc480535420)

[Interview with End User 3](#_Toc480535421)

[User Needs 3](#_Toc480535422)

[User Objectives 4](#_Toc480535423)

[Limitations 4](#_Toc480535424)

[Evidence of Analysis 4](#_Toc480535425)

[Success Criteria 5](#_Toc480535426)

[Proposed Solutions 6](#_Toc480535427)

[Other Pondered Solutions 6](#_Toc480535428)

[Design 7](#_Toc480535429)

[Classes 7](#_Toc480535430)

[Algorithms 7](#_Toc480535431)

[Work Flow Diagram 9](#_Toc480535432)

[Testing 10](#_Toc480535433)

[Networks 10](#_Toc480535434)

[Strings 11](#_Toc480535435)

[Lists 12](#_Toc480535436)

[Matrices 12](#_Toc480535437)

[Other 13](#_Toc480535438)

[Evaluation 14](#_Toc480535439)

[Personal Evaluation 14](#_Toc480535440)

[End User Evaluation 14](#_Toc480535441)

[Appendix - Code 15](#_Toc480535442)

[Main.py 15](#_Toc480535443)

[BasicFunctionsLibrary.py 16](#_Toc480535444)

[Ciphers.py 17](#_Toc480535445)

[Hashing.py 20](#_Toc480535446)

[Matrices.py 21](#_Toc480535447)

[Networks.py 23](#_Toc480535448)

[Nim.py 27](#_Toc480535449)

[Sorting.py 29](#_Toc480535450)

[SpanningTree.py 30](#_Toc480535451)

[StatisticsCheatSheet.py 32](#_Toc480535452)

# Analysis

Introduction

In Computer Science and Decision Maths there are many algorithms and methods that are used. They are often for data management such as sorting and searching but networks are another large part of the courses.

Current System

Currently when teaching certain algorithms and theories, there are no working examples in Python which can be shown to the students. The teachers only options are to have the students creating their own code from a loose understanding or use with animations no accessible code behind them.

The teacher currently has no example algorithms which can be used in lessons when teaching key algorithms or when practising techniques such as Object Oriented programming or recursive programming. The only way is to explain the algorithms in English and then have the students work out the algorithms on their own. This is good as it forces the students to really understand the fundamentals; however, there are situations when examples will help or when the fundamentals are already there and something more complex is being worked on.

End User

For this project, my end user is the head of Computer Science, Mr Roush.

Interview with End User

**In class what do you currently use for teaching algorithms?**

Currently, we only use the book and some YouTube clips to explain the algorithm and then the students will code it.

**What programming language do you currently use?**

For general purpose, we use Python however SQL is also used when learning about databases.

**So if they’re stuck, is there a fall back?**

Usually, if someone is stuck the students will work together and most of the time as a group they can work through the problem. However this can lead to some members of the group falling behind, and if none of them can I will have to code my own example.

**Would an algorithm library be useful?**

On the occasions where they have tried but need a hint, a look through of a working bit of code, with the ability to test the answers, would be very useful. Also a library could be imported and sections used in large programs if the understanding was already there.

User Needs

Mr Roush needs a library of algorithms which are all annotated thoroughly so that they can be used as examples and followed through. They need to be not only easy to test but also accessible to be imported to other programs.

User Objectives

I aim to have:

* A simple text interface.
* The entire project will be written in Object Oriented Programming so that I can learn object oriented programming and it is a scalable system
* Examples of
  + Sorting algorithms.
  + A Search algorithm.
  + Dijkstra’s Algorithm and another example of route finding algorithms.
  + Matrices operations such as addition and inversion.
  + Ciphers and Hashing algorithms.
  + Spanning Algorithms (possibly for Decision Maths so Kruskal’s and Primm’s).
  + Any other useful or fun algorithms I can find to fit into the library.
* Accessible code to all users

Limitations

There are a few limitations, given or otherwise. One is the library which will be a Python library as this is what primarily is used in the classroom and it has great versatility. Also I’ve decided to limit it to a purely text based interface as with a library the main feature is the code and a GUI would be used rarely and only when testing the algorithms. So the users will primarily use the Python IDE for accessing and manipulating the code.

Another limitation is that though each algorithm should be able to be called independently, they won’t work on their own. This is because certain functions will be shared throughout similar algorithms. Therefore the entire program will needed for it to run nominally. This means if a user wants to import it they’ll either have to import the entire program or transplant the algorithm.

Evidence of Analysis

**Libraries**

To make a library which can be easily accessed and exported from, I’m going to make the entire project in different files to differentiate different sections and keep all the algorithms within their own classes so importing is easy and takes everything required.

**Inverting Matrices**

To make an inverting algorithm I first had to learn how to hand inverse n x n matrices. I originally found a relatively easy, logic heavy, technique called “Row Operations Inversion”. A second is the a method of using the determinant. For this I had to evaluate which would be a better system to emulate in code:

|  |  |
| --- | --- |
| Row Operations Method | Determinant Method |
| {    {    {    {    {    Therefore the Inverse of A :  ... a dumb man came to shout hooray. A deaf policeman heard the noise and came killed the two dead boys, a paralysed donkey walking by kicked the copper in the eye, into a dry ditch and killed the all. Don’t believe me this lie is true ask the blind man he saw it two! | |  |  | | --- | --- | | Example | Technique | | Finding the determinant: |  | |  | A(EI-FH)-B(DI-FG)+C(DH-EG) | | Finding the cofactors matrix: |  | |  | To find the cofactors matrix for each position remove that row and column and find determinant of remaining and changing the signs by row swapping each time. | | Put them together: |  | |  | The inverse matrix is simply the one over the determinant times the cofactors matrix flipped along the diagonal axis from (0,0 )to (n,n).  One fine day in the middle of the night, two dead men got up to fight, back to back they faced each other, drew their swords and shot each other. One was blind and the other couldn’t see so they got a dummy for a referee, a blind man came to see fair play … | |
| This technique may be easier to do as a human but this is because it uses a lot of logic, in other words it’s not algorithmic and getting a computer to do the correct operations would be incredibly difficult and it would only lose a few operations. | This technique requires a lot more maths so takes longer for a human however it has the advantage of being algorithmic and code be far more easily converted into a set of instructions for a computer. |

Success Criteria

* The Main Page should:
  + Separate algorithms by complexity.
  + Auto return after algorithm is complete or if there’s been an error.
  + Not be required for algorithms to run.
* Networks should:
  + Have simple system of entering graphs.
  + Have the possibility to copy previously used graphs straight in.
  + Have one centralised class for the input/output of graphs.
  + Have inheritance for shared functions.
  + Show the times of different shortest path algorithms.
  + Be able to handle both integers and floats.
* Strings should:
  + Have an example of a Hashing algorithm.
  + Be able to hash and un-hash the results.
  + Have encryption and decryption.
  + Have some sort of simple decryption for when the key is unknown, as an example.
  + Have the possibility of both types of the Vigenère cipher, where the key is the same length as the phrase and when it’s not.
  + Have various examples of making the ciphers more complex.
* Lists should:
  + Have multiple examples of sorting algorithms, definitely quick sort.
  + Preferably have a good and bad example of a sorting algorithm.
  + Be able to handle both integers and floats.
* Matrices should:
  + Have both types of matrix multiplication.
  + Have a working matrix inversion algorithm.
  + Be able to inverse n x n matrices.
  + Be able to handle both integers and floats.
* Other criteria are:
  + Have a text interface.
  + Having no persistent errors when running it normally.
  + Being robust enough not to crash when being used incorrectly.
  + That all the algorithms are at least within methods and can be exported to other programs.
  + There must be an impossible game!

Proposed Solutions

My aim is to build an extensive code library which can be used so students can see how the algorithm is translated into Python, in a way which also allows them to test it with a range of inputs. I also aim for it to be easy imported so they can be used in conjunction with each other or even in other programs.

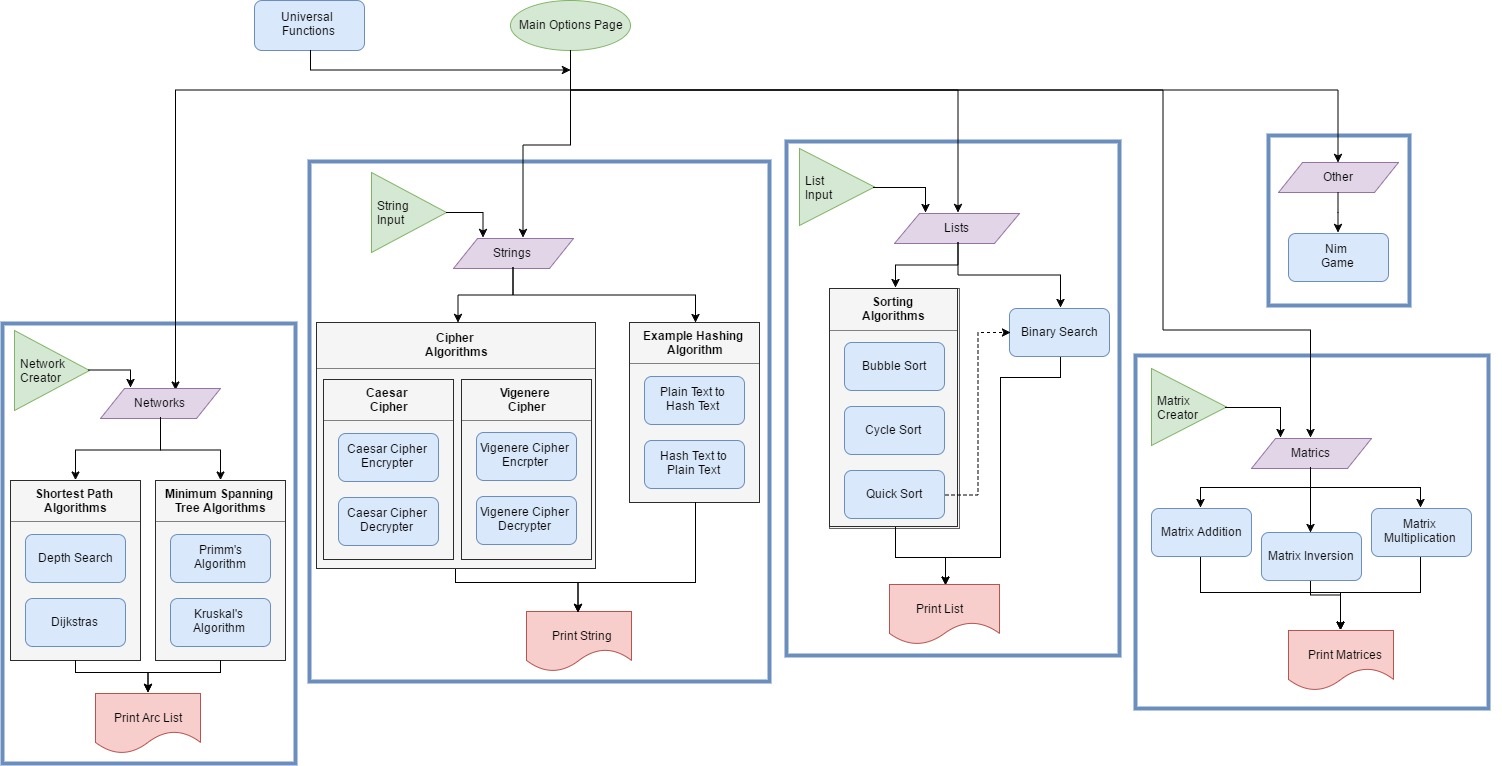
Other Pondered Solutions

One addition to the idea was to build a website around the Python code so it could be tested online and then downloaded but this was considered overcomplicating it and as the school already have a good secure intranet, the code would only be moved over there for easier use.

A second proposal was to create a visual animation of the algorithms, using Python or more likely MATLAB. There were a few problems with this idea, one was the likely learning of a new language which was difficult as I knew no one who knew the language and online tutorials where often expensive. A secondary issue and more important was it didn’t really fix the issue as animations already existed, though rarely with editable data, and it was the recreating of the algorithms in Python afterwards that could cause problems so having working algorithms was more important.

# Design

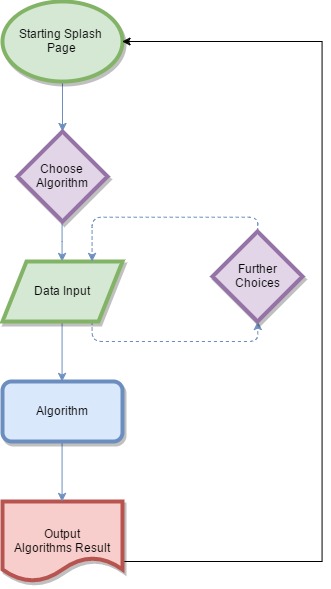
Classes



Algorithms

|  |
| --- |
| **Matrix Inversion:**  After my analysis on the two methods, I decided to use the Determinant method for finding the inverse as it’s far more algorithmic and though the calculations are more longwinded this is something computers excel at far more than humans.  Pseudo Code:  *Input : Matrix*  *Function removeAdjacent (tempMatrix,X,Y):*  *remove tempMatrix[Y]*  *for count = 0 to len(tempMatrix):*  *remove tempMatrix[i][X]*  *return tempMatrix*  *Function getDeterminant(Matrix):*  *if len(Matrix[0]) == 1:*  *return Matrix[0][0]*  *elif len(Matrix[0]) == 2:*  *return Matrix[0][0]\*Matrix[1][1]-Matrix[0][1]\*Matrix[1][0]*  *else:*  *tempNum = 0*  *for rNum = 1 to len(Matrix[0]):*  *subMatrix = removeAdjacent(copyof(Matrix),rNum,0)*  *subSumDeter = GetDeterminant(subMatrix)*  *tempNum = tempNum + (1-2\*(rNum%2))\*Matrix[0][rNum]\*subSumDeter*  *return tempNum*    *Function matrixInverser(Matrix):*  *determinant = getDeterminant(Matrix)*  *MatrixN = createBlankMatrix(n,n)*  *for x = 0 to N:*  *for y = 0 to N:*  *subMatrix = removeAdjacent(copyof(Matrix),x,y)*  *numN = getDeterminant(subMatrix)*  *MatrixN[x][y] = (1/determiannt)\*(1-2\*((xLoc+yLoc)%2))\*numN*  *return MatrixN*  *Output : matrixInverser(Matrix)* |
| **Dijkstra’s:**  As networks are a key aspect of computer science, finding the shortest route is a common question and Dijkstra’s shortest path algorithms is one of the best, non-heuristic, algorithm for finding it.  Pseudo Code:  Inputs : Matrix, StartLoc,TargetLoc  Global Variables(StartLoc,RouteStore,DistanceStore,Matrix)  DistanceStore = [0]\*len(Matrix)  RouteStore = [""]\*len(Matrix)  Function GetRoute(TargetLoc,TraversalList):  TraversalList.append(TargetLoc)  for point0 = 0 to len(DistanceStore):  if StartLoc == TargetLoc:  return TraversalList  elif DistanceStore[TargetLoc] == DistanceStore[point0] + Matrix[TargetLoc][point0] and point0 != TargetLoc:  return GetRoute(point0,TraversalList)  Function DijkstraMain(CurrentLoc,TargetLoc,PassNo):  RouteStore[CurrentLoc] = PassNo  for point1 = 0 to len(Matrix):  TravelDistance = Matrix[CurrentLoc][point1] + DistanceStore[CurrentLoc]  if (DistanceStore[point1] > TravelDistance or DistanceStore[point1] == 0) and RouteStore[point1] == "" and Matrix[CurrentLoc][point1] != 0:  DistanceStore[point1] = TravelDistance  LowestValue = 0  NextLoc = "Limit Reached"  for point2 = 0 to len(Matrix):  if RouteStore[point2] == "" and DistanceStore[point2] != 0 and (DistanceStore[point2] < LowestValue or LowestValue == 0) :  LowestValue = DistanceStore[point2]  NextLoc = point2  if NextLoc == TargetLoc:  return GetRoute(TargetLoc,[])  elif NextLoc == "Limit Reached" :  return GetRoute(TargetLoc,[])  else:  return DijkstraMain(NextLoc,TargetLoc,PassNo+1)    Output : DijkstraMain(StartLoc,TargetLoc,0) |
| **Cycle Sort:**  Another even more common data type are lists, and so often they need to be ordered. There are countless sorting algorithms all of varying degrees of efficiency, all with their pros and cons. This algorithm, Cycle sort, is a particularly interesting unstable sorting algorithm.  Pseudo Code:  Input : List  for cycle = 0 to len(List):  pos = cycle + 1  while pos != cycle:  pos = cycle  for index = List[cycle+1:]:  if index < List[cycle]:  pos = pos + 1  if pos != cycle:  while List[cycle] == List[pos]:  pos = pos + 1  List[pos], List[cycle] = List[cycle], List[pos]  Output : List |
| **Example Hashing:**  Hashing Algorithms have become incredibly important with advent of the internet. As they are realistically secure way of testing information. Such as for error correction or to keep sites secure. Due to the nature of hashing algorithms there are an infinite amount of combinations for hashing algorithms, this is just one example.  Pseudo Code:  *Input : PlainText*  *itemList = list(PlainText)*  *hashList = []*  *for i = 0 to len(itemList):*  *itemNo = {ASCII}.index(itemList[i])*  *hashList.append(itemNo\*(i+1))*  *hash = "".join(hashList)*  *Output : hash*  An improvement could be of looping the date, or other inputs to further mix the numbers, into multiplication line. |
| **Impossible Nim AI:**  Nim is game with the aim of not reaching zero. It starts by picking a random starting number and then each player has to minus a number of one to three. This is done repeatedly until one person is forced to choose a number that will lead to zero, so losing them the game.  Pseudo Code:  *Input : stackValue*  *choiceRing = [3,randomInteger(1,3),1,2]*  *choiceCalc = stackValue % 4*  *choice = choiceRing[choiceCalc]*  *Output : choice*  This algorithm is so simple but providing the stack doesn’t start as , it will win every time. |

Work Flow Diagram

* When the program is run from the main.py file it will start with a text interface, this will have a options page which sends you to the algorithm of your choice.
* Once there you will input your data and add any further parameters for the algorithms (eg. Type of sorting algorithm).
* The algorithm will run.
* The output will be printed and then the program will loop back to the original options page from main.py.

# Testing

This table shows the evidence for completing the success criteria.

|  |  |  |
| --- | --- | --- |
| **The** Main Page **should:** |  | |
| Auto return to (or at least the previous option) after algorithm is complete or if there’s been an error. | Separate algorithms by complexity. | |
| Not be required for algorithms to run. | |
| Networks **should:** |  | |
| Have simple system of entering graphs. | Have the possibility to copy previously used graphs straight in. | |
| Have inheritance for shared functions. | |
| Have one centralised class for the input/output of graphs. | Show the times of different shortest path algorithms. | |
| Be able to handle both integers and floats. | |
| Strings **should:** |  | |
| Have an example of a Hashing algorithm. | | Be able to hash and un-hash the results. |
| Have encryption and decryption for all the ciphers. | | Have some sort of simple decryption for when the key is unknown, as an example. |
| Have the possibility of both types of the Vigenère cipher, where the key is the same length as the phrase and when it’s not. | | Have examples of making the ciphers more complex. |
| Lists **should:** |  | |
| Have multiple examples of sorting algorithms, definitely quick sort. | Preferably have a good and bad example of a sorting algorithm. | |
| Be able to handle both integers and floats. | |
| Matrices **should:** |  | |
| Have both types of matrix multiplication. | Be able to inverse n x n matrices. | |
| Be able to handle both integers and floats. | Have a working matrix inversion algorithm. | |
| Other **criteria are:** |  | |
| Having no persistent errors when running it normally.  None have been found in the extensive testing. | That all the algorithms are at least within methods and can be exported to other programs.  Look to the code appendix. | |
| There must be an impossible game!  The starting stack is any 4n+c ()except 4n+1, this because the algorithm is supposed to work in multiples of 4, so matching the user so it goes down in fours until it AI is left with C so it can leave the player with one left in the stack.  The impossible version has the AI start and the stack never be 4n+1. | Being robust enough not to crash when being used incorrectly. | |
| Have a text interface. |

# Evaluation

Personal Evaluation

I set out to build a library of algorithms and in my opinion, I’ve satisfactorily completed this aim. It was a pretty open ended project, at least in what algorithms I was going to include. This I think benefitted the project hugely because it meant after I’d added the most important, Dijkstra’s and Quicksort particularly, I had free reign to add as many and as varied algorithms as I found interesting. For example, Cyclesort is a rarely used sorting algorithm but it uses a very different technique. This I found interesting and my project allowed me to add it with the other sorting algorithms I did.

An issue I found with the freeness for me to choose algorithms was it was difficult to draw a line on where to stop as apart from the main ones used in the lessons, which were added first and relatively quickly as I had already learnt them, I could keep adding algorithms indefinitely, making a more and more extensive library. I am happy with the number I have, as there are many varied examples and operations in each section.

I think if I were to do this again, I would want to make a GUI as though it was unnecessary for a code library. On the odd occasions when large networks or matrices were needed to be added. It was simple but it took a long time and a GUI would hugely improve this. I wouldn’t want to use tkinter though and I would probably use PHP with python backend, if possible, for testing and then have a download for when you want to use the library. This not only adds to the ease of use but also the accessibility of the code as it could all be run on an embedded website on the school intranet.

End User Evaluation

In terms of an end-user evaluation, I feel like it meets all stated success criteria. The user interface is a bit clunky, but all text-based interfaces are. Takes me back to computing circa 1990, when “point-and-click” was a bit of a buzzword! In general, this is not a limitation, but it becomes slightly problematic when dealing with the graph algorithms.

It is thoroughly robust. I intentionally entered bogus data a few times, and the program would return to the main menu without encountering a runtime error. I like the way sorting algorithms are dealt with. Most importantly, it didn’t crash when asked to invert a 4 x 2 matrix.

I do wish it would provide a “show your working” level of detail on some of the algorithms. It shows output, but not some steps in between. Especially when it comes to sorting or hashing, this should not be difficult to do. For the graph algorithms, I totally accept that this would be more than too much.

I like the range of algorithms provided. They are not necessarily all related, but it is nice to have such a varied menu of math ‘toys’ to play with.

### I had some time before the project was handed, so I added some working for the sorting and hashing algorithm.

# Appendix - Code

Main.py

*import* Nim *as* nimGame  
*import* Networks *as* network  
*import* Ciphers *as* cipher  
*import* BasicFunctionsLibrary *as* bfl  
*import* StatisticsCheatSheet *as* scs  
*import* Matrices *as* mat  
*import* Hashing *as* hash  
  
### This is the main class which when you are testing the algorithms runs everything.  
  
*class* Runner():  
  
 *def* \_\_init\_\_(self):  
 self.choice = 0  
  
 ### This is function that contains the loop which continuously keeps the program running.  
  
 *def* Start(self):  
 self.Welcome()  
 *while True*:  
 self.Choose()  
 self.RunChoice()  
  
 ### These two functions are primarily text and tell the user all the options and takes their choice as an input.  
  
 *def* Welcome(self):  
 print("\n-=-=-=-=-=-=-=-=-=-= Welcome to the A level Coding Library =-=-=-=-=-=-=-=-=-=-\n\nThis is a library of algorithms of varying complexity which you will most likely\nuse in this course.")  
 print("")  
  
 *def* Choose(self):  
 print("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-\n\nPlease choose one of the various algorithms to run/inspect:")  
 print("\n-=-=-=-= Easy =-=-=-=-")  
  
 print("(1) The BubbleSort Algorithm")  
 print("(2) The Caesar Cipher")  
  
 print("\n-=-=-=- Medium -=-=-=-")  
  
 print("(3) Primm's Algorithm")  
 print("(4) Kruskal's Algorithm")  
 print("(5) Vigenere Cipher")  
 print("(6) Example Hashing Algorithm")  
 print("(7) The CycleSort Algorithm")  
  
 print("\n-=-=-=-= Hard =-=-=-=-")  
  
 print("(8) Dijkstra's Algorithm")  
 print("(9) The QuickSort Algorithm")  
 print("(10) Matrix Operations")  
 print("(11) Binomial Recursion")  
 print("(12) Recursive Depth Search")  
  
 print("\n-=-=-=- Extras -=-=-=-")  
  
 print("(13) The Nim Game")  
 print("(14) Compare Depth Search and Dijkstra's")  
 print("(15) Statistics Cheat Sheet")  
  
 print("\n\nTo choose please enter the corresponding index :",end=" ")  
 self.choice = input()  
  
 ### This function takes the input from the overarching Start function and then sends the user to the appropriate algorithm.  
  
 *def* RunChoice(self):  
 *try*:  
 *if* self.choice == "1":  
 network.Initialise(self.choice)  
 *elif* self.choice == "2":  
 Caesar = cipher.CaesarBot()  
 Caesar.Initialise()  
  
 *elif* self.choice == "3":  
 network.Initialise(self.choice)  
 *elif* self.choice == "4":  
 network.Initialise(self.choice)  
 *elif* self.choice == "5":  
 Vigenere = cipher.VigenereBot()  
 Vigenere.Initialise()  
 *elif* self.choice == "6":  
 Hasher = hash.HashBot()  
 Hasher.Initialise()  
 *elif* self.choice == "7":  
 network.Initialise(self.choice)  
  
  
 *elif* self.choice == "8":  
 network.Initialise(self.choice)  
 *elif* self.choice == "9":  
 network.Initialise(self.choice)  
 *elif* self.choice == "10":  
 Matrix = mat.MatrixBot()  
 Matrix.Initialise()  
 *elif* self.choice == "11":  
 bfl.BinomialDistribution()  
 *elif* self.choice == "12":  
 network.Initialise(self.choice)  
  
 *elif* self.choice == "13":  
 nimGame.Initialise()  
 *elif* self.choice == "14":  
 network.Initialise(self.choice)  
 *elif* self.choice == "15":  
 scs.Start()  
  
 *elif* self.choice == "66":  
 Secret = cipher.SecretCipher()  
 Secret.Initialise()  
  
 *else*:  
 print("\nInput Error ---- [Restarting]")  
 int("爆発!")  
 *except*:  
 print("\nError Occurred - [ErrorLoc : Main]")  
 *pass*### Starts everything!  
  
Alpha = Runner()  
Alpha.Start()

BasicFunctionsLibrary.py

### This error checks networks to confirm that the data is all correct.  
  
*def* NetworkChecker(*matrix*):  
 ErrorList = []  
 Correct = *True* NoVertices = len(*matrix*)  
 *for* y *in* range(NoVertices):  
 *for* x *in* range(NoVertices):  
 #print("Checking:",matrix[x][y],"+",matrix[y][x])  
 *if matrix*[x][y] != *matrix*[y][x]:  
 Correct = *False* ErrorList.append(str(y)+","+str(x))  
 *return* Correct  
  
### This function returns all the arcs from a network.  
  
*def* GetArcListFromNetwork(*Matrix*):  
 ArcList = []  
 *for* y *in* range(len(*Matrix*)):  
 *for* x *in* range(len(*Matrix*)):  
 *if Matrix*[y][x] != 0 *and* x > y:  
 ArcList.append([y,x])  
 *return* ArcList  
  
### This functions gets the mode of the list.  
  
*def* HighestOccurrence(*list*,*target*):  
 store = 0  
 TopLoc = 0  
 *for* i *in* range(len(*list*)):  
 No = *list*[i].count(*target*)  
 *if* No > store:  
 store = No  
 TopLoc = i  
 *return* TopLoc  
  
### This is a recursive algorithm for calculating binomial distributions to a specific order.  
  
*def* BinomialDistribution():  
 *def* FreakingRecursion(*Row*):  
 *if Row* == 0:  
 *return* [1]  
 *else*:  
 lastRow = FreakingRecursion(*Row* - 1)  
 thisRow = []  
 *for* index *in* range(0, len(lastRow) + 1):  
 *if* index == 0 *or* (len(lastRow)) == index:  
 thisRow.append(1)  
 *else*:  
 Value = lastRow[index] + lastRow[index - 1]  
 thisRow.append(Value)  
 *return* thisRow  
  
 order = int(input("\nEnter the order of binomial you want: "))  
 PascalsList = FreakingRecursion(order - 1)  
  
 print(PascalsList)  
  
### This removes an item from a list and then shifts the rest of the list back to fill in the space.  
  
*def* delListShift(*List*,*target*):  
 tLoc = *List*.index(*target*)  
 *for* point *in* range(len(*List*)):  
 *if* point > tLoc:  
 *List*[point-1] = *List*[point]  
 *List*.pop()  
 *return List*

Ciphers.py

*import* BasicFunctionsLibrary *as* bfl  
*import* string  
*import* random  
  
### This is the class for the Caesar cipher.  
  
*class* CaesarBot():  
  
 *def* \_\_init\_\_(self):  
 self.phrase = ""  
 self.pushNo = 0  
  
 ### The initializer simply gets the user inputs and choice and sends it to processing.  
  
 *def* Initialise(self):  
 print("\n\n-=-=-=-=-=-=-=-=-=-= The Caesar Cipher =-=-=-=-=-=-=-=-=-=-\n")  
 print("This is a very simple cipher which shifts the letters by a certain an to decode only reverse shifting is required.\n")  
  
 Stay = *True  
 while* Stay:  
 *try*:  
 choice = input("\nDo you want to encode(1), decode(2) or crack(3) a message (leave blank to return to main menu) : ")  
 *if* choice == "":  
 Stay = *False  
  
 elif* choice == "1":  
 self.phrase = input("\nEnter phrase to encode : ")  
 self.pushNo = input("Enter the push number : ")  
 print("""Encoding " """+self.phrase+""" " \n\tpushed by""",self.pushNo+".")  
 print("\nThe enciphered version of this is :",self.EnDeCoder(int(self.pushNo)))  
 print("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-")  
  
 *elif* choice == "2":  
 self.phrase = input("\nEnter phrase to decode : ")  
 self.pushNo = input("Enter the pull number : ")  
 print("""Decoding " """+self.phrase+""" " \n\tpulled by""",self.pushNo+".")  
 print("\nThe deciphered version of this is :",self.EnDeCoder(-int(self.pushNo)))  
 print("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-")  
  
 *elif* choice == "3":  
 self.phrase = input("\nEnter phrase to decode : ")  
 print("Deciphering",self.phrase,":\n")  
 self.GaiusCassiusLonginus()  
  
 *else*:  
 int("This won't work - ;)")  
  
 *except*:  
 print("\nInput Error ---- [Restarting Choice]")  
  
 ### This function is used to decode text when the key isn't known, it uses frequency and though is relatively simple it is pretty accurate so illustrating the shortcomings of the caesar cipher.  
  
 *def* GaiusCassiusLonginus(self):  
 print("\tAll possible plain versions(in order of likelihood):")  
 self.phrase = self.phrase.lower()  
 PhraseOptions = []  
 LetterFrequency = ['e', 't', 'a', 'o', 'i', 'n', 's', 'r', 'h', 'l', 'd', 'c', 'u', 'm', 'f', 'p', 'g', 'w', 'y', 'b', 'v', 'k', 'x', 'j', 'q', 'z']  
 *for* brutus *in* range(1,26):  
 PhraseOptions.append(self.EnDeCoder(-brutus).lower())  
 *for* marcus *in* range(25):  
 TopValueIndex = bfl.HighestOccurrence(PhraseOptions,LetterFrequency[marcus])  
 print("\t("+str(marcus+1)+")",PhraseOptions[TopValueIndex])  
 *del* PhraseOptions[TopValueIndex]  
 print("\n\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-\n")  
  
 ### This is the encoder which is what shifts the plain text, so encoding it. To reverse the same function is used just with a negative shift value.  
  
 *def* EnDeCoder(self,*ppNo*):  
 List = list(self.phrase)  
 *for* point *in* range(len(List)):  
 *try*:  
 List[point] = string.ascii\_letters[(list(string.ascii\_letters).index(List[point])+*ppNo*)%52]  
 *except*:  
 *pass  
 return* "".join(List)  
  
### This is the class for the Vigenere cipher.  
  
*class* VigenereBot():  
  
 *def* \_\_init\_\_(self):  
 self.plainTxt = ""  
 self.cipherTxt = ""  
  
 ### The low and high level functions set what characters are available to be encoded and they include just letter characters and all of the ascii characters respectively.  
  
 *def* LowLevel(self):  
 self.unitList = list(string.ascii\_letters)  
 self.unitList.append(" ")  
  
 *def* HighLevel(self):  
 self.unitList = list(string.printable)  
  
 ### This simply gets the user input for the key or if none is given randomly generates its own depending on the encoding level.  
  
 *def* GetCipherKey(self):  
 *if* self.cipherTxt == "":  
 cipherStore = []  
 self.cipherTxt = []  
 *for* i *in* range(len(self.plainTxt)):  
 randIndex = random.randint(0,len(self.unitList)-1)  
 self.cipherTxt.append(randIndex)  
 cipherStore.append(self.unitList[randIndex])  
 *else*:  
 cipherStore = list(self.cipherTxt)[:len(self.plainTxt)]  
 self.cipherTxt = []  
 *for* i *in* range(len(cipherStore)):  
 self.cipherTxt.append(self.unitList.index(cipherStore[i]))  
 *return* "".join(cipherStore)  
  
 ### The encoder and decoder are separate in this cipher as the key is stored in character form which doesn't include negatives.  
  
 *def* Encoder(self,*plainTxt*,*key*):  
 *for* point *in* range(len(*plainTxt*)):  
 *try*:  
 *plainTxt*[point] = self.unitList[(self.unitList.index(list(*plainTxt*)[point])+*key*[point%len(*key*)])%len(self.unitList)]  
 *except*:  
 *pass  
 return* "".join(*plainTxt*)  
  
 *def* Decoder(self,*cipherTxt*,*key*):  
 *for* point *in* range(len(*cipherTxt*)):  
 *try*:  
 *cipherTxt*[point] = self.unitList[(self.unitList.index(list(*cipherTxt*)[point])-*key*[point%len(*key*)])%len(self.unitList)]  
 *except*:  
 *pass  
 return* "".join(*cipherTxt*)  
  
 ### The initializer primary input is how you want to run the program but it also takes the data.  
  
 *def* Initialise(self):  
 print("\n\n-=-=-=-=-=-=-=-=-=-= The Vigenere Cipher =-=-=-=-=-=-=-=-=-=-\n")  
 print("This is an incredibly secure cipher particularly if the key is the same length as the pass phrase.\n")  
  
 Stay = *True  
 while* Stay:  
 *try*:  
 choiceB = int(input("\nDo you want to encode(1) or decode(2) a message [To leave enter (0)] : "))  
 *if* choiceB == 0:  
 Stay = *False  
 else*:  
 choiceA = input("\n\t~ To start enter the level of encryption : High(1) or Low(2) : ")  
 *if* choiceA == "1":  
 self.HighLevel()  
 *elif* choiceA == "2":  
 self.LowLevel()  
 *else*:  
 int("This won't work")  
   
 *if* choiceB == 1:  
 self.plainTxt = input("\nEnter phrase to encode : ")  
 self.cipherTxt = input("Enter the key(leave blank to have one generated) : ")  
 print("""Encoding " """+self.plainTxt+""" " \n\twith key {""",self.GetCipherKey(),"}.")  
 print("\nThe enciphered version of this is :",self.Encoder(list(self.plainTxt),self.cipherTxt))  
 print("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-")  
  
 *elif* choiceB == 2:  
 self.plainTxt = input("\nEnter phrase to decode : ")  
 self.cipherTxt = input("Enter the key(leave blank to have one generated) : ")  
 print("""Decoding " """ + self.plainTxt + """ " \n\twith key""", self.GetCipherKey())  
 print("\nThe deciphered version of this is :", self.Decoder(list(self.plainTxt),self.cipherTxt))  
 print("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-")  
  
 *else*:  
 int("This won't either work - ;)")  
  
 *except*:  
 print("\nInput Error ---- [Restarting Choice]")  
  
### This a secret, shh!  
  
*class* SecretCipher():  
  
 *def* \_\_init\_\_(self):  
 self.vowels = ['a', 'A', 'e', 'E', 'i', 'I', 'o', 'O', 'u', 'U', 'y', 'Y']  
 self.consonants = ['b','B', 'c','C', 'd','D', 'f','F', 'g','G', 'h','H', 'j','J', 'k','K', 'l','L', 'm','M', 'n','N', 'p','P', 'q','Q', 'r','R', 's','S', 't','T', 'v','V', 'w','W', 'x','X', 'z','Z']  
  
 *def* Initialise(self):  
 print("\n\n-=-=-=-=-=-=-=-=-=-= The Language Generator =-=-=-=-=-=-=-=-=-=-")  
 print("\nThis is will mix consonants and vowels depending on a specific key")  
  
 self.key = input("\n\t~Enter the language key(leave blank to have one generated) : ")  
 print("\t\t~Your key is",self.GetKey())  
 self.ConvertAlphabet()  
  
 Stay = *True  
 while* Stay:  
 *try*:  
 phrase = str(input("\nPlease enter your phrase to translate(leave empty to return to main menu) : "))  
 *if* phrase == "":  
 Stay = *False  
 pass  
 else*:  
 print(self.Translator(phrase))  
 *except*:  
 print("\nInput Error ---- [Restarting Input]\n")  
 *pass  
  
  
 def* Translator(self,*phrase*):  
 PhraseToList = list(*phrase*)  
 TranslatedPhrase = []  
 *for* current *in* PhraseToList:  
 *if* current *in* self.vowels:  
 TranslatedPhrase.append(self.ConvertedVowels[self.vowels.index(current)])  
 *elif* current *in* self.consonants:  
 TranslatedPhrase.append(self.ConvertedConsonants[self.consonants.index(current)])  
 *else*:  
 TranslatedPhrase.append(current)  
 *return* "".join(TranslatedPhrase)  
  
  
  
 *def* GetKey(self):  
 *if* self.key == "":  
 cipherStore = []  
 self.key = []  
 *for* i *in* range(random.randint(1,12)):  
 randIndex = random.randint(0,len(string.ascii\_letters)-1)  
 self.key.append(randIndex)  
 cipherStore.append(string.ascii\_letters[randIndex])  
 *else*:  
 cipherStore = list(self.key)  
 self.key = []  
 *for* i *in* range(len(cipherStore)):  
 self.key.append(string.printable.index(cipherStore[i]))  
 *return* "".join(cipherStore)  
  
 *def* ConvertAlphabet(self):  
 self.ConvertedVowels = []  
 self.ConvertedConsonants = []  
  
 *for* indexA *in* range(len(self.vowels)//2):  
 letter = self.vowels[2\*((indexA+self.key[indexA%len(self.key)])%6)]  
 self.ConvertedVowels.append(letter)  
 self.ConvertedVowels.append(letter.upper())  
  
 *for* indexB *in* range(len(self.consonants)//2):  
 letter = self.consonants[2\*((indexB+self.key[indexB%len(self.key)])%20)]  
 self.ConvertedConsonants.append(letter)  
 self.ConvertedConsonants.append(letter.upper())

Hashing.py

*import* string  
  
### This is the class which is an example hashing algorithm.  
  
*class* HashBot():  
  
 *def* \_\_init\_\_(self):  
 self.item = "Strings. Strings. Everywhere!"  
 print("\n\n-=-=-=-=-=-=-=-=-=-= An Example Hashing Algorithm =-=-=-=-=-=-=-=-=-=-")  
  
 ### The initializer function as usual takes the users choice and inputs and hands the off to the other functions in the object.  
  
 *def* Initialise(self):  
 Stay = *True  
 while* Stay:  
 *try*:  
 self.item = input("\nEnter text : ")  
  
 print("Do you want to run a hash with working(1) or without working(2) \nor run an unhash with working(3) or without working(4) on this text [To leave enter (0)] :",end=" ")  
  
 choice = int(input())  
 *if* choice == 1:  
 print("Hashed Text :",self.makeHashWorking())  
 *elif* choice == 2:  
 print("Hashed Text :",self.makeHash())  
 *elif* choice == 3:  
 print("Plain Text :",self.undoHashWorking())  
 *elif* choice == 4:  
 print("Plain Text :",self.undoHash())  
 *elif* choice == 0:  
 Stay = *False  
 else*:  
 int("This message will self-destruct now!")  
 *except*:  
 print("\nInput Error ---- [Restarting Choice]")  
 self.Initialise()  
  
 ### This turns regular plain text into a simple hash.  
  
 *def* makeHash(self):  
 itemList = list(self.item)  
 hashList = []  
 *for* i *in* range(len(itemList)):  
 itemNo = string.printable.index(itemList[i])  
 hashList.append(itemNo\*(i+1))  
 print("")  
 *return* " ".join(str(x) *for* x *in* hashList)  
  
 ### This is the opposite so turns the hash back into plain text.  
  
 *def* undoHash(self):  
 itemList = []  
 hashList = self.item.split()  
 *for* i *in* range(len(hashList)):  
 itemNo = int(hashList[i])  
 itemList.append(string.printable[int(itemNo/(i+1))])  
 print("")  
 *return* "".join(itemList)  
  
 ### This turns regular plain text into a simple hash, however also shows the working.  
  
 *def* makeHashWorking(self):  
 itemList = list(self.item)  
 hashList = []  
 *for* i *in* range(len(itemList)):  
 itemNo = string.printable.index(itemList[i])  
 hashList.append(itemNo\*(i+1))  
 print("\nIn position",i+1,"is",itemList[i],"the ASCII number for this is",itemNo," \n\t\ttherefore the hash number is",i+1,"x",itemNo,"=",str(itemNo\*(i+1))+".")  
 print("")  
 *return* " ".join(str(x) *for* x *in* hashList)  
  
 ### This is the opposite so turns the hash back into plain text, however also shows the working.  
  
 *def* undoHashWorking(self):  
 itemList = []  
 hashList = self.item.split()  
 *for* i *in* range(len(hashList)):  
 itemNo = int(hashList[i])  
 itemList.append(string.printable[int(itemNo/(i+1))])  
 print("\nIn position",i+1,"is",hashList[i],"so to find the ASCII number,",hashList[i],"/",str(i+1)+", this number is equivalent to",string.printable[int(itemNo/(i+1))]+".")  
 print("")  
 *return* "".join(itemList)

Matrices.py

*import* copy  
  
### This is the class that contains all the matrices operations algorithms.  
  
*class* MatrixBot():  
  
 *def* \_\_init\_\_(self):  
 *pass* ### This initializer only gets the choice of operator from the user then passes them to the appropriate function.  
  
 *def* Initialise(self):  
 print("\n\n-=-=-=-=-=-=-=-=-=-= Matrices =-=-=-=-=-=-=-=-=-=-\n")  
 print("Do you want to add(1), multiply(2) or inverse(3) a matrix : ",end="")  
 *try*:  
 choice = int(input())  
 *if* choice == 1:  
 sumMatrix = self.matrixAdder()  
 self.PrintMatrix(sumMatrix)  
 *elif* choice == 2:  
 multiMatrix = self.matrixMultiplierInitialiser()  
 self.PrintMatrix(multiMatrix)  
 *elif* choice == 3:  
 InvMatrix = self.matrixInverserComplex()  
 self.PrintMatrix(InvMatrix)  
 *else*:  
 int("Hqak ak kiddikeg hi wbeew : Bombarda")  
 *except*:  
 *return* self.Initialise()  
  
  
 ### Adds two matrices of the same dimension!  
  
 *def* matrixAdder(self):  
 x,y = self.GetDimensions()  
 MatrixA = self.CreateMatrix(x,y)  
 self.PrintMatrix(MatrixA)  
 MatrixB = self.CreateMatrix(x,y)  
 self.PrintMatrix(MatrixB)  
 MatrixNew = self.CreateBlank(x,y)  
 *for* yLoc *in* range(y):  
 *for* xLoc *in* range(x):  
 MatrixNew[yLoc][xLoc] = MatrixA[yLoc][xLoc] + MatrixB[yLoc][xLoc]  
 *return* MatrixNew  
  
  
##### Inverting n x n Matrices !  
  
 ### This takes a matrix and removes all values on the same row and column of a position, then returns the matrix.  
  
 *def* removeAdjacent(self,*tempMatrix*,*X*,*Y*):  
 *del tempMatrix*[*Y*]  
 *for* i *in* range(len(*tempMatrix*)):  
 *del tempMatrix*[i][*X*]  
 *return tempMatrix* ### This function works out the determinant of a matrix.  
  
 *def* GetDeterminant(self,*Matrix*):  
 *if* len(*Matrix*[0]) == 1:  
 *return Matrix*[0][0]  
 *elif* len(*Matrix*[0]) == 2:  
 *return Matrix*[0][0]\**Matrix*[1][1]-*Matrix*[0][1]\**Matrix*[1][0]  
 *else*:  
 tempNum = 0  
 *for* rNum, rItem *in* enumerate(*Matrix*[0]):  
 tempNum += (1-2\*(rNum%2))\*rItem\*self.GetDeterminant(self.removeAdjacent(copy.deepcopy(*Matrix*),rNum,0))  
 *return* tempNum  
  
 ### This is the function that, using the above functions, works out the inverse matrix from the user input.  
  
 *def* matrixInverserComplex(self):  
 print("\nPlease enter matrix dimensions (n x n) :")  
 N = int(input("N : "))  
 Matrix = self.CreateMatrix(N,N)  
 self.PrintMatrix(Matrix)  
 determinant = self.GetDeterminant(Matrix)  
 print("\n\tThe matrix determinant was",str(determinant)+".")  
 MatrixN = self.CreateBlank(N,N)  
  
 *for* yLoc *in* range(N):  
 *for* xLoc *in* range(N):  
 MatrixN[yLoc][xLoc] = (1/determinant)\*(1-2\*((xLoc+yLoc)%2))\*self.GetDeterminant(self.removeAdjacent(copy.deepcopy(Matrix),yLoc,xLoc))  
 *return* MatrixN  
  
  
##### Matrix multiplication!  
  
 ### This is effectively the initializer for matrix multiplication, so it takes the users choice and runs it.  
  
 *def* matrixMultiplierInitialiser(self):  
 print("\n\tPlease enter whether you want to multiply two matrices(1) or multiply a matrix by a constant(2) :",end=" ")  
 *try*:  
 choice = int(input())  
 *if* choice == 1:  
 *return* self.matrixMultiplierComplex()  
 *elif* choice == 2:  
 *return* self.matrixMultiplierSimple()  
 *else*:  
 int("ផ្ទុះ")  
 *except*:  
 *return* self.matrixMultiplierInitialiser()  
  
 ### Multiplies a matrix by a constant.  
  
 *def* matrixMultiplierSimple(self):  
 *try*:  
 x,y = self.GetDimensions()  
 Matrix = self.CreateMatrix(x,y)  
 num = float(input("Enter number to multiply :"))  
 self.PrintMatrix(Matrix)  
 *for* yLoc *in* range(y):  
 *for* xLoc *in* range(x):  
 Matrix[yLoc][xLoc] = Matrix[yLoc][xLoc]\*num  
 *return* Matrix  
 *except*:  
 *return* self.matrixMultiplierSimple()  
  
 ### Multiplies two matrices, however is mainly just a passing function taking user input to the maths function.  
  
 *def* matrixMultiplierComplex(self):  
 x,y = self.GetDimensions()  
 MatrixA = self.CreateMatrix(x,y)  
 self.PrintMatrix(MatrixA)  
 MatrixB = self.CreateMatrix(y,x)  
 self.PrintMatrix(MatrixB)  
  
 *return* self.MatrixComp(x,y,MatrixA,MatrixB)  
  
 ### This is the function that actually multiplies the two matrices and is called by above function.  
  
 *def* MatrixComp(self,*minVal*,*maxVal*,*HighMatrix*,*LowMatrix*):  
 MatrixM = self.CreateBlank(*minVal*,*minVal*)  
 *for* A *in* range(len(MatrixM)):  
 *for* B *in* range(len(MatrixM)):  
 temp = 0  
 *for* add *in* range(*maxVal*):  
 temp += *HighMatrix*[add][B]\**LowMatrix*[A][add]  
 MatrixM[A][B] = temp  
 *return* MatrixM  
  
##### ----------------------------------------------------------------------------------------------------------------  
  
 ### This creates an n x n identity matrix.  
  
 *def* CreateIdentity(self,*n*):  
 Matrix = self.CreateBlank(*n*,*n*)  
 *for* i *in* range(*n*):  
 Matrix[i][i] = 1  
 *return* Matrix  
  
 ### This creates an X x Y blank matrix.  
  
 *def* CreateBlank(self,*X*,*Y*):  
 blank = []  
 *for* a *in* range(*Y*):  
 temp = []  
 *for* b *in* range(*X*):  
 temp.append(0)  
 blank.append(temp)  
 *return* blank  
  
 ### This function prints matrices.  
  
 *def* PrintMatrix(self,*Matrix*):  
 print("")  
 *for* a *in* range(len(*Matrix*)):  
 print("| ",end="")  
 *for* b *in* range(len(*Matrix*[a])-1):  
 print(round(*Matrix*[a][b],3),end=", ")  
 print(round(*Matrix*[a][len(*Matrix*[a])-1],3),"|")  
 print("")  
  
 ### A simple function for getting user input of the dimensions.  
  
 *def* GetDimensions(self):  
 *try*:  
 print("\nPlease enter matrix dimensions :")  
 x = int(input("X : "))  
 y = int(input("Y : "))  
 *return* x,y  
 *except*:  
 *return* self.GetDimensions()  
  
 ### This gets user input and turns it into a correctly formatted matrix for the other functions to use.  
  
 *def* CreateMatrix(self,*x*,*y*):  
 Matrix = self.CreateBlank(*x*,*y*)  
 print("Now enter the matrix items(row then collumn)[{},{}]".format(*x*,*y*))  
 *for* yLoc *in* range(*y*):  
 *for* xLoc *in* range(*x*):  
 output = "Please enter item at point ("+str(xLoc)+","+str(yLoc)+") (leave blank for zero): "  
 *try*:  
 Matrix[yLoc][xLoc] = float(input("Please enter item at point ({},{}) (leave blank for zero): ".format(xLoc+1,yLoc+1)))  
 *except*:  
 *pass  
 return* Matrix

Networks.py

*import* Sorting *as* sort  
*import* BasicFunctionsLibrary *as* bfl  
*import* SpanningTree *as* st  
*import* Hashing *as* hash  
*import* time  
  
### This is a secondary function for running which starts all the algorithms which use the class NetworkBot, it's called directly from Main.  
  
*def* Initialise(choice):  
 ClassBotAlpha = NetworkBot()  
 *if* choice == "1":  
 print("\n\n-=-=-=-=-=-=-=-=-=-= The BubbleSort Algorithm =-=-=-=-=-=-=-=-=-=-\n")  
 print(sort.BubbleBot(ClassBotAlpha.MakeList()))  
 *elif* choice == "3":  
 Matrix, NodeNames = ClassBotAlpha.MakeNetwork()  
 Primms = st.PrimmBot(Matrix,NodeNames)  
 Primms.Initialise()  
 *elif* choice == "4":  
 Matrix, NodeNames = ClassBotAlpha.MakeNetwork()  
 Kruskal = st.KruskalBot(Matrix,NodeNames)  
 Kruskal.Initialise()  
 *elif* choice == "7":  
 print("\n\n-=-=-=-=-=-=-=-=-=-= The CycleSort Algorithm =-=-=-=-=-=-=-=-=-=-\n")  
 CycleSortBot = sort.CycleBot(ClassBotAlpha.MakeList())  
 print("Sorted list:",CycleSortBot.Initialise())  
 *elif* choice == "8":  
 Dijkstra = DijkstraBot(0,0)  
 Dijkstra.Initialise()  
 *elif* choice == "9":  
 print("\n\n-=-=-=-=-=-=-=-=-=-= The QuickSort Algorithm =-=-=-=-=-=-=-=-=-=-\n")  
 QuickSortBot = sort.QuickBot(ClassBotAlpha.MakeList())  
 print("\nSorted list:",QuickSortBot.Initialise())  
 *elif* choice == "12":  
 DepthSearch = DepthBot(0,0)  
 DepthSearch.Initialise()  
 *elif* choice == "14":  
 Matrix, NodeNames = ClassBotAlpha.MakeNetwork()  
 OverFunc = ShortestPathBot(Matrix, NodeNames)  
 start, end = OverFunc.Input()  
 timeDi = DijkstraBot(Matrix, NodeNames)  
 timeDe = DepthBot(Matrix, NodeNames)  
 timeDi.Timer(start,end)  
 timeDe.Timer(start,end)  
  
 *else*:  
 print("\nError Occurred ---- [ErrorLoc : Networks]")  
  
### This is a binary search algorithm.  
  
*def* BinarySearch(*List*,*Target*):  
 quick = sort.QuickBot(*List*)  
 orderedList = quick.Initialise()  
 center = (len(orderedList)+1)//2  
 *if Target* == orderedList[center]:  
 *return True  
 elif* len(orderedList) <= 2:  
 *return False  
 elif Target* > orderedList[center]:  
 *return* BinarySearch(orderedList[len(orderedList)//2:],*Target*)  
 *elif Target* < orderedList[center]:  
 *return* BinarySearch(orderedList[:len(orderedList)//2],*Target*)  
  
### This is the class which primarily does the Network handling but also the list handling.  
  
*class* NetworkBot():  
  
 *def* \_\_init\_\_(self):  
 *pass* ### This is the function that gets a list from the user.  
  
 *def* MakeList(self):  
 print("Please enter your list [separate the values, floats only, with spaces]")  
 print("\nList :",end=" ")  
 *try*:  
 List = [float(item) *for* item *in* input().split()]  
 *except*:  
 print("\nInput Error ---- [Restarting Choice]")  
 List = self.MakeList()  
 newList = []  
 *for* cur *in* List:  
 *if* cur%1 == 0:  
 newList.append(int(cur))  
 *else*:  
 newList.append(cur)  
 *return* newList  
  
 ### This function is that allows a user to directly paste in previously used lists.  
  
 *def* GetFullNetwork(self):  
 *try*:  
 network = input("List : ")  
 tempList = network[2:-2].split("], [")  
 newList = []  
 *for* i *in* range(len(tempList)):  
 Atemp = tempList[i].split(", ")  
 Btemp = []  
 *for* x *in* range(len(Atemp)):  
 Btemp.append(float(Atemp[x]))  
 newList.append(Btemp)  
 print(newList)  
 *if* bfl.NetworkChecker(newList):  
 *return* newList, len(newList[0])  
 *else*:  
 *return* self.GetFullNetwork()  
 *except*:  
 *return* self.GetFullNetwork()  
  
 ### This simply gets the names of each node in the network.  
  
 *def* NameList(self,*NodeNo*):  
 print("")  
 NodeNames = []  
 *for* currentNode *in* range(*NodeNo*):  
 print("Enter the name of node",currentNode+1,end=" : ")  
 NodeNames.append(input())  
 *return* NodeNames  
  
 ### This is the network creation function.  
  
 *def* MakeNetwork(self):  
 network = []  
 print("\n\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-\n")  
 print("To create your network, please enter the number of nodes in your graph")  
 *try*:  
 NodeNo = int(input("\nNumber of nodes[Leave blank to paste whole network] : "))  
 *except*:  
 print("\nPlease enter the complete list(formatted the same as when copied)")  
 network, NodeNo = self.GetFullNetwork()  
 *return* network, self.NameList(NodeNo)  
  
 *for* a *in* range(NodeNo):  
 temp = []  
 *for* b *in* range(NodeNo):  
 temp.append(0)  
 network.append(temp)  
  
 NodeNames = self.NameList(NodeNo)  
  
 *for* x *in* range(NodeNo-1):  
 print("\nPlease enter the weight of the nodes connecting to",NodeNames[x],"[If there is no connection leave blank]\n")  
 *for* y *in* range(NodeNo):  
 *if* x < y:  
 print("The weight of connection between",NodeNames[x],"and",NodeNames[y],end=" : ")  
 *try*:  
 currentWeight = float(input())  
 network[x][y] = currentWeight  
 network[y][x] = currentWeight  
 *except*:  
 *pass* bfl.NetworkChecker(network)  
 print("Network created :",network,"\n\n")  
 *return* network, NodeNames  
  
### This class is the overarching class for the two shortest path algorithms.  
  
*class* ShortestPathBot(NetworkBot):  
  
 *def* \_\_init\_\_(self,*network*,*NodeName*):  
 NetworkBot.\_\_init\_\_(self)  
 self.algorName = "Timer"  
 *if network* == 0 *and NodeName* == 0: # Gets the network if one has not been given #  
 self.network, self.NodeNames = NetworkBot.MakeNetwork(self)  
 *else*:  
 self.network = *network* self.NodeNames = *NodeName* ### This starts the algorithm section and gets the start and end locations.  
  
 *def* Input(self):  
 print("\n-=-=-=-=-=-=-=-=-=-=",self.algorName,"=-=-=-=-=-=-=-=-=-=-\n")  
 print("To start the algorithm on your network please enter a start and end location.\n")  
 *for* i *in* range(len(self.network[0])):  
 print("["+str(self.NodeNames[i]),"=",str(i)+"]",end=" ")  
 print("")  
 *try*:  
 startLoc = int(input("Start Location(NodeNumber) : "))  
 TargetLoc = int(input("End Location(NodeNumber) : "))  
 *except*:  
 startLoc, TargetLoc = self.Input()  
 *return* startLoc, TargetLoc  
  
 ### Outputs the list in a linear english form.  
  
 *def* returnArc(self,*List*):  
 print(self.NodeNames[*List*[0]],end="")  
 *del List*[0]  
 *for* i *in List*:  
 print(" to",self.NodeNames[i],end="")  
 print("")  
  
 ### This function returns the sum of the given list of arcs.  
  
 *def* SumList(self,*List*):  
 sum = 0  
 *for* i *in* range(len(*List*)-1):  
 sum += self.network[*List*[i+1]][*List*[i]]  
 *return* sum  
  
 ### This outputs a given list of arcs using the two above functions.  
  
 *def* Output(self,*List*):  
 print("\n\n\t∴ Shortest path :",end=" ")  
 self.returnArc(*List*[:])  
 print("\t\t~ of weight :",self.SumList(*List*))  
  
### This class is for Dijkstra's Algorithm.  
  
*class* DijkstraBot(ShortestPathBot):  
  
 *def* \_\_init\_\_(self,*network*,*NodeName*):  
 ShortestPathBot.\_\_init\_\_(self,*network*,*NodeName*)  
 self.algorName = "Dijkstra's Algorithm"  
 self.DistanceStore = [0]\*len(self.network)  
 self.RouteStore = [""]\*len(self.network)  
  
 ### This functions calls all the other functions in order for Dijkstra to run properly.  
  
 *def* Initialise(self):  
 StartLoc, TargetLoc = self.Input()  
 self.StartStore = StartLoc  
 self.FinalList = self.DijkstraMain(StartLoc,TargetLoc,0)  
 self.FinalList.reverse()  
 self.Output(self.FinalList)  
  
 ### This function similar to the above function except it measures the run time.  
  
 *def* Timer(self,*StartLoc*,*TargetLoc*):  
 timerStart = time.time()  
 self.StartStore = *StartLoc* self.FinalList = self.DijkstraMain(*StartLoc*,*TargetLoc*,0)  
 timerEnd = time.time()  
 self.FinalList.reverse()  
 self.Output(self.FinalList)  
 print("\tDijkstra Time :",timerEnd-timerStart)  
  
 ### This works back through the network to find the shortest route after being given the traversal list.  
  
 *def* GetRoute(self,*TargetLoc*,*TraversalList*):  
 *TraversalList*.append(*TargetLoc*)  
 *for* point0 *in* range(len(self.DistanceStore)):  
 *if* self.StartStore == *TargetLoc*:  
 *return TraversalList  
 elif* self.DistanceStore[*TargetLoc*] == self.DistanceStore[point0] + self.network[*TargetLoc*][point0] *and* point0 != *TargetLoc*:  
 *return* self.GetRoute(point0,*TraversalList*)  
  
 ### This the traversal list which is the shortest route from the start node to all the other nodes.  
  
 *def* DijkstraMain(self,*CurrentLoc*,*TargetLoc*,*PassNo*):  
 self.RouteStore[*CurrentLoc*] = *PassNo* ### Gets working value for each of the connected locations.  
 *for* point1 *in* range(len(self.network)):  
 TravelDistance = self.network[*CurrentLoc*][point1] + self.DistanceStore[*CurrentLoc*]  
 *if* (self.DistanceStore[point1] > TravelDistance *or* self.DistanceStore[point1] == 0) *and* self.RouteStore[point1] == "" *and* self.network[*CurrentLoc*][point1] != 0:  
 self.DistanceStore[point1] = TravelDistance  
  
 ### Determines the next current location.  
 LowestValue = 0  
 NextLoc = "Limit Reached"  
 *for* point2 *in* range(len(self.network)):  
 *if* self.RouteStore[point2] == "" *and* self.DistanceStore[point2] != 0 *and* (self.DistanceStore[point2] < LowestValue *or* LowestValue == 0) :  
 LowestValue = self.DistanceStore[point2]  
 NextLoc = point2  
  
 ### Checks to see if target reached then passes to next iteration of the code.  
 *if* NextLoc == *TargetLoc*:  
 *return* self.GetRoute(*TargetLoc*,[])  
 *elif* NextLoc == "Limit Reached" :  
 *return* self.GetRoute(*TargetLoc*,[])  
 *else*:  
 *return* self.DijkstraMain(NextLoc,*TargetLoc*,*PassNo*+1)  
  
### This is the class for the depth search algorithm.  
  
*class* DepthBot(ShortestPathBot):  
  
 *def* \_\_init\_\_(self,*network*,*NodeName*):  
 ShortestPathBot.\_\_init\_\_(self,*network*,*NodeName*)  
 self.algorName = "Depth Search Algorithm"  
  
 ### This functions starts the depth search and then finds the shortest of the found routes.  
  
 *def* Initialise(self):  
 StartLoc, self.TargetLoc = self.Input()  
 self.foundRoutes = []  
 self.RecurSearch([StartLoc])  
 shortest = self.foundRoutes[0]  
 *for* current *in* self.foundRoutes:  
 *if* self.SumList(current) < self.SumList(shortest):  
 shortest = current  
 self.Output(shortest)  
  
 ### This function similar to the above function except it measures the run time.  
  
 *def* Timer(self,*StartLoc*,*TargetLoc*):  
 timerStart = time.time()  
 self.TargetLoc = *TargetLoc* self.foundRoutes = []  
 self.RecurSearch([*StartLoc*])  
 shortest = self.foundRoutes[0]  
 *for* current *in* self.foundRoutes:  
 *if* self.SumList(current) < self.SumList(shortest):  
 shortest = current  
 timerEnd = time.time()  
 self.Output(shortest)  
 print("\tDepth Time :",timerEnd-timerStart)  
  
 ### Returns all the arcs connected to a given node.  
  
 *def* GetConnections(self,*currentLoc*):  
 ConnectedArcs = []  
 ArcList = bfl.GetArcListFromNetwork(self.network)  
 *for* point *in* range(len(ArcList)):  
 *if currentLoc in* ArcList[point]:  
 ConnectedArcs.append(ArcList[point])  
 *return* ConnectedArcs  
  
 ### This recursively looks through each possible route, which doesn't cycle, and then stores it.  
  
 *def* RecurSearch(self,*NodeList*):  
 possArcs = self.GetConnections(*NodeList*[len(*NodeList*)-1])  
 *for* i *in* possArcs:  
 *if* i[0] *in NodeList and* i[1] *in NodeList*:  
 possArcs.remove(i)  
  
 *for* i *in* range(len(possArcs)):  
 y, x = possArcs[i]  
 *if* self.TargetLoc == x *or* self.TargetLoc == y:  
 *NodeList*.append(self.TargetLoc)  
 self.foundRoutes.append(*NodeList*[:])  
 *elif not*(x *in NodeList*):  
 nextList = *NodeList*[:]  
 nextList.append(x)  
 self.RecurSearch(nextList)  
 *elif not*(y *in NodeList*):  
 nextList = *NodeList*[:]  
 nextList.append(y)  
 self.RecurSearch(nextList)

Nim.py

*import* random  
  
### This class is for the stack.  
  
*class* Pile():  
  
 *def* \_\_init\_\_(self,*currentValue*):  
 self.currentValue = *currentValue  
  
 def* removeAmount(self,*removeAmount*):  
 self.currentValue = self.currentValue - *removeAmount*### This class is for the AI.  
  
*class* AIPlayer():  
  
 *def* \_\_init\_\_(self,*aiName*,*currentChoice*):  
 self.aiName = *aiName* self.currentChoice = *currentChoice* ### This function returns the AI choice on medium difficulty, it returns a 3:1 random mix of the hard and the easy choices.  
  
 *def* MakeChoiceMed(self,*TheStack*):  
 RandMed = random.randint(0,100)  
  
 *if* RandMed < 75:  
 self.currentChoice = self.MakeChoiceEasy()  
 *else*:  
 self.currentChoice = self.MakeChoiceHard(*TheStack*)  
  
 *return* self.currentChoice  
  
 ### This function returns the AI choice on easy difficulty, it returns a random choice.  
  
 *def* MakeChoiceEasy(self):  
 self.currentChoice = random.randint(1,3)  
 *return* self.currentChoice  
  
 ### This function returns the AI choice on hard difficulty, it uses the maths of the game to calculate the perfect answer and will always work providing the stack doesn't start as 4n + 1.  
  
 *def* MakeChoiceHard(self,*TheStack*):  
 choiceRing = [3,random.randint(1,3),1,2]  
 choiceCalc = *TheStack*.currentValue % 4  
  
 self.currentChoice = choiceRing[choiceCalc]  
 *return* self.currentChoice  
  
### This is the class for the player.  
  
*class* HumanPlayer():  
  
 *def* \_\_init\_\_(self,*playerName*,*currentChoice*):  
 self.PlayerName = *playerName* self.currentChoice = *currentChoice* ### This function only gets the users choice and returns it.  
  
 *def* MakeChoice(self):  
 NotDone = *True  
 while* NotDone:  
 print(self.PlayerName,"make your choice(1-3) : ")  
 self.currentChoice = int(input())  
  
 *if* self.currentChoice >= 1 *and* self.currentChoice <= 3:  
 NotDone = *False  
  
 return* self.currentChoice  
  
### This is the function that runs the game.  
  
*def* GameStart(*GameType*,*TheStack*):  
  
 ### This runs the game with two players.  
  
 *if GameType* == 1:  
 aName = input("\nPlayer 1 choose name : ")  
 bName = input("Player 2 choose name : ")  
 PlayerA = HumanPlayer(aName,0)  
 PlayerB = HumanPlayer(bName,0)  
 RUN = *True  
  
 while* RUN:  
  
 *if* RUN:  
 print("\nThe current value is",*TheStack*.currentValue)  
 *TheStack*.removeAmount(PlayerA.MakeChoice())  
  
 *if TheStack*.currentValue <= 0:  
 WhoLose = PlayerA.PlayerName  
 RUN = *False  
  
 if* RUN:  
 print("\nThe current value is",*TheStack*.currentValue)  
 *TheStack*.removeAmount(PlayerB.MakeChoice())  
  
 *if TheStack*.currentValue <= 0:  
 WhoLose = PlayerB.PlayerName  
 RUN = *False* ### This runs the game with a player and an AI.  
  
 *elif GameType* >= 2:  
 *if GameType* == 5: # Sets the stack for impossible to be unbeatable (removes the possibility of it ever being 4n + 1). #  
 List = [16,18,19,20,22,23,24]  
 *TheStack*.currentValue = List[random.randint(0,2)]  
 Name = input("\nPlayer choose name : ")  
 Player = HumanPlayer(Name,0)  
 aiPlayer = AIPlayer("NimBot",0)  
  
 RUN = *True  
  
 while* RUN:  
  
 *if* RUN:  
 print("\nThe current value is",*TheStack*.currentValue)  
 TempStore = *TheStack*.currentValue  
  
 *if GameType* == 2: # This is where the difficulty is decided #  
 *TheStack*.removeAmount(aiPlayer.MakeChoiceEasy())  
 *elif GameType* == 3:  
 *TheStack*.removeAmount(aiPlayer.MakeChoiceMed(*TheStack*))  
 *elif GameType* == 4 *or GameType* == 5:  
 *TheStack*.removeAmount(aiPlayer.MakeChoiceHard(*TheStack*))  
 *else*:  
 print("Error : Code 1")  
  
 print(aiPlayer.aiName,"chose",TempStore-*TheStack*.currentValue)  
  
 *if TheStack*.currentValue <= 0:  
 WhoLose = aiPlayer.aiName  
 RUN = *False  
  
 if* RUN:  
 print("\nThe current value is",*TheStack*.currentValue)  
 *TheStack*.removeAmount(Player.MakeChoice())  
  
 *if TheStack*.currentValue <= 0:  
 WhoLose = Player.PlayerName  
 RUN = *False* print(WhoLose,"loses!")  
  
### This gets the user input for the game type and then runs the appropriate type.  
  
*def* Initialise():  
 print("\n\n-=-=-=-=-=-=-=-=-=-= Welcome to Nim =-=-=-=-=-=-=-=-=-=-\n\n")  
 print("The game is too force your opposition to be the one to make \nzero by choosing the correct numbers.")  
  
 print("\nTo start, please choose gamemode :")  
 print("\n (1) Player vs Player")  
 print(" (2) Player vs Computer[Easy]")  
 print(" (3) Player vs Computer[Medium]")  
 print(" (4) Player vs Computer[Hard]")  
 print(" (5) Player vs Computer[Impossible]")  
 print("\n (0) Choose a different algorithm")  
  
 *try*:  
 print("\n\nTo choose please enter the corresponding index :",end=" ")  
 choice = input()  
  
 *if* choice == "1" *or* choice == "2" *or* choice == "3" *or* choice == "4" *or* choice == "5":  
 TheStack = Pile(random.randint(15,25))  
 GameStart(int(choice),TheStack)  
 *elif* choice == "0":  
 *pass  
 else*:  
 int("This better break ;)")  
  
 *except*:  
 print("\nInput Error ---- [Restarting Nim]")  
 Initialise()

Sorting.py

### This is the function which does bubble sort.  
  
*def* BubbleBot(*List*):  
 *for* passNo *in* range(len(*List*)-1,0,-1):  
 *for* index *in* range(passNo):  
 *if List*[index]>*List*[index+1]:  
 tempItem = *List*[index]  
 *List*[index] = *List*[index+1]  
 *List*[index+1] = tempItem  
 *return List*### This is the class for Cycle Sort.  
  
*class* CycleBot():  
  
 *def* \_\_init\_\_(self,*List*):  
 self.List = *List* ### This is the function which actually does the sort.  
  
 *def* Initialise(self):  
 *for* cycLoc *in* range(len(self.List)): # Runs through each item in list #  
 pos = cycLoc + 1  
 *while* pos != cycLoc:  
 pos = cycLoc  
 *for* index *in* self.List[cycLoc+1:]: # Searches through rest of the list to see how many are items are greater than the cycLoc #  
 *if* index < self.List[cycLoc]:  
 pos += 1  
 *if* pos != cycLoc:  
 *while* self.List[cycLoc] == self.List[pos]: # Checks that the location found is not already occupied by a value equal to the original #  
 pos += 1  
 self.List[pos], self.List[cycLoc] = self.List[cycLoc], self.List[pos] # Swaps the original with the new point #  
  
 *return* self.List  
  
### This is the class for quick sort.  
  
*class* QuickBot():  
  
 *def* \_\_init\_\_(self,*List*):  
 self.List = *List* ### This function only starts the computing function.  
  
 *def* Initialise(self):  
 toWork = input("Enter (0) for working otherwise leave blank: ")  
 *return* self.QuickRecur(self.List,toWork)  
  
  
  
 ### This is the computing function for quick sort.  
  
 *def* QuickRecur(self,*currentList*,*working*):  
 *if* len(*currentList*) <= 1:  
 *return currentList  
 else*:  
 pivot = *currentList*[len(*currentList*)//2] # Finds the pivot for the list #  
 ListX = []  
 ListE = []  
 ListN = []  
 *for* index *in currentList*: # This for works through each item in the list and assigns it the correct list #  
 *if* index > pivot:  
 ListX.append(index) # For values greater than the pivot #  
 *elif* index == pivot:  
 ListE.append(index) # For values equal to the pivot #  
 *else*:  
 ListN.append(index) # For values less than the pivot #  
  
 *if working* == "0":  
 print("\nFor list:",*currentList*,"with pivot",ListE[0])  
 ListT = [ListN,ListE,ListX]  
 print("Splits into |",end=" ")  
 *for* num *in* range(3):  
 *if* len(ListT[num]) > 0:  
 print(ListT[num],"|",end=" ")  
 print("")  
  
 *return* self.QuickRecur(ListN,*working*) + ListE + self.QuickRecur(ListX,*working*) # This returns the lists in order but also recursively orders those lists independently #

SpanningTree.py

*import* BasicFunctionsLibrary *as* bfl  
  
### This is the class for Primm's algorithm.  
  
*class* PrimmBot():  
  
 *def* \_\_init\_\_(self,*Matrix*,*NodeNames*):  
 self.Matrix = *Matrix* self.NodeNames = *NodeNames* print("\n-=-=-=-=-=-=-=-=-=-= Primm's Algorithm =-=-=-=-=-=-=-=-=-=-\n")  
 *for* i *in* range(len(self.Matrix[0])):  
 print("["+str(self.NodeNames[i]),"=",str(i)+"]",end=" ")  
 print("")  
  
 ### Gets user input for starting point then runs Primm's algorithm througth the other functions.  
  
 *def* Initialise(self):  
 *try*:  
 self.NodeList = [int(input("\nPlease enter the number of the starting point : "))]  
 print("")  
 *except*:  
 self.Initialise()  
 *while* len(self.NodeList) != len(self.NodeNames):  
 nextArc = self.GetNextArc()  
 *for* i *in* range(2):  
 *if not* nextArc[i] *in* self.NodeList:  
 self.NodeList.append(nextArc[i])  
 print("Add arc of weight",self.Matrix[nextArc[0]][nextArc[1]],"between nodes",self.NodeNames[nextArc[0]],"and",self.NodeNames[nextArc[1]],"to the network")  
  
 ### This looks through all the arcs and finds the lowest one, which won't create a loop, to return.  
  
 *def* GetNextArc(self):  
 curConArcs = []  
 *for* i *in* range(len(self.NodeList)):  
 curConArcs.extend(self.GetConnections(self.NodeList[i]))  
 possArcs = self.RemovePairs(curConArcs)  
  
 MinArc = "One fine day in the middle of the night!"  
 *for* point *in* range(len(possArcs)):  
 y, x = possArcs[point]  
 *if not*(y *in* self.NodeList *and* x *in* self.NodeList):  
 *if* MinArc == "One fine day in the middle of the night!":  
 MinArc = self.Matrix[y][x]  
 currArc = [y,x]  
 *elif* self.Matrix[y][x] < MinArc:  
 MinArc = self.Matrix[y][x]  
 currArc = [y,x]  
 *return* currArc  
  
 ### Removes pairs from a list.  
  
 *def* RemovePairs(self,*List*):  
 *for* i *in* range(len(*List*)):  
 *try*:  
 *if List*[i] *in List*[:i]:  
 dupe = *List*[i]  
 List = bfl.delListShift(bfl.delListShift(*List*,dupe),dupe)  
 *except*:  
 *pass  
 return List* ### Returns all the arcs connecting to the given node.  
  
 *def* GetConnections(self,*currentLoc*):  
 ConnectedArcs = []  
 ArcList = bfl.GetArcListFromNetwork(self.Matrix)  
 *for* point *in* range(len(ArcList)):  
 *if currentLoc in* ArcList[point]:  
 ConnectedArcs.append(ArcList[point])  
 *return* ConnectedArcs  
  
### This is the class for Kruskal's Algorithm.  
  
*class* KruskalBot():  
  
 *def* \_\_init\_\_(self,*Matrix*,*NodeNames*):  
 self.Matrix = *Matrix* self.NodeNames = *NodeNames* self.OrderedList = []  
  
 ### This function checks to see if a loop would be created if the arc between A and B would be added to the new network.  
  
 *def* LoopChecker(self,*checkList*,*pointA*,*pointB*,*attempts*):  
 *if pointA* == *pointB*:  
 self.GlobalBoolean = *False  
 elif attempts* != len(*checkList*):  
 *for* index *in* range(len(*checkList*)):  
 *try*:  
 currentLoc = *checkList*[index].index(*pointA*)  
 self.LoopChecker(*checkList*,*checkList*[index][1-currentLoc],*pointB*,*attempts*+1)  
 *except*:  
 *pass* ### This is the function that runs the algorithm using the other functions and then prints the results.  
  
 *def* ArcAdder(self):  
 checkList = []  
 *for* count *in* range(len(self.OrderedList)):  
 y, x = self.OrderedList[count]  
 self.GlobalBoolean = *True* self.LoopChecker(checkList,self.OrderedList[count][0],self.OrderedList[count][1],0)  
 *if* self.GlobalBoolean:  
 print("Add arc of weight",self.Matrix[y][x],"between nodes",self.NodeNames[self.OrderedList[count][0]],"and",self.NodeNames[self.OrderedList[count][1]],"to the network")  
 checkList.append(self.OrderedList[count])  
  
 ### This function puts the arc in order it in terms of weight, using a system similar to bubble sort.  
  
 *def* Positioner(self,*current*,*checkLoc*):  
 y, x = self.ArcList[*current*]  
 yp, xp = self.OrderedList[*checkLoc*]  
 *if* self.Matrix[y][x] >= self.Matrix[yp][xp]:  
 self.OrderedList.insert(*checkLoc*+1,self.ArcList[*current*])  
 *elif checkLoc* == 0:  
 self.OrderedList.insert(*checkLoc*,self.ArcList[*current*])  
 *else*:  
 self.Positioner(*current*,*checkLoc*-1)  
  
 ### This function gets the arc list and puts them in a list, using the above function to order it.  
  
 *def* GetArcsIndex(self):  
 self.ArcList = bfl.GetArcListFromNetwork(self.Matrix)  
  
 *for* count *in* range(len(self.ArcList)):  
 *if* count == len(self.ArcList):  
 *return* self.OrderedList  
 *elif* count == 0:  
 self.OrderedList.append(self.ArcList[count])  
 *else*:  
 self.Positioner(count, len(self.OrderedList)-1)  
  
 ### This sets of the algorithm.  
  
 *def* Initialise(self):  
 print("\n\n-=-=-=-=-=-=-=-=-=-= Kruskal's Algorithm =-=-=-=-=-=-=-=-=-=-\n")  
 self.GetArcsIndex()  
 print("The arcs in order :",self.OrderedList,"\n")  
 self.ArcAdder()

StatisticsCheatSheet.py

*import* statistics  
*import* math  
  
### This is their because it is useful, not because it's a good example of anything. <--- Do you think I should keep this, Sir?  
  
*def* FreakingCoding(*calcList*):  
 codedList = []  
 additionFactor = float(input("Enter number to add(0 for no change) : "))  
 divideNumber = float(input("Enter number to divide(1 for no change) : "))  
 *for* i *in* range(len(*calcList*)):  
 codedList.append((*calcList*[i]+additionFactor)/divideNumber)  
 *return* codedList  
  
*def* MidCorrerlater(*list*):  
 squareList = []  
 *for* a *in* range(len(*list*)):  
 squareList.append(*list*[a]\*\*2)  
 ans = sum(squareList)-(sum(*list*))\*\*2/len(*list*)  
 *return* ans  
  
*def* Correrlater(*calcList*):  
 print("Enter second list :")  
 newList = GetList()  
 Scc = MidCorrerlater(*calcList*)  
 Snn = MidCorrerlater(newList)  
 additionList = []  
 *for* V *in* range(len(*calcList*)):  
 additionList.append(newList[V]\**calcList*[V])  
 Snc = sum(additionList)-(sum(*calcList*)\*sum(newList))/len(*calcList*)  
 print("Scc :",round(Scc,1))  
 print("Snn :",round(Snn,1))  
 print("Snc :",round(Snc,1))  
  
 print("Correlation Coefficient :",round(Snc/math.sqrt(Scc\*Snn),3))  
  
*def* PointPercentile(*calcList*):  
 Pct = int(input("\nEnter percentile point : "))  
 size = len(*calcList*)  
 *return* sorted(*calcList*)[int(math.ceil((size \* Pct) / 100)) - 1]  
  
*def* QuartileRange(*type*,*calcList*):  
 rangeSize = math.ceil(len(*calcList*)/4)  
 *if type* == 0:  
 *return* QuartileRange(3,*calcList*) - QuartileRange(1,*calcList*)  
 *else*:  
 *return* sorted(*calcList*)[int(math.ceil((len(*calcList*) \* (*type*/ 4))))-1]  
  
  
*def* GetList():  
 *return* [float(item) *for* item *in* input().split()]  
  
*def* Variance(*calcList*):  
 squareList = []  
 *for* a *in* range(len(*calcList*)):  
 squareList.append(*calcList*[a]\*\*2)  
 σσ = (sum(squareList)/len(squareList))-(sum(*calcList*)/len(*calcList*))\*\*2  
 *return* σσ  
  
*def* Start():  
 *try*:  
 print("\n\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-\n")  
 print("\nEnter the list of numbers(space to seperate) :")  
 StartMain(GetList())  
 *except*:  
 print("\nInput Error ---- [Restarting Choice]")  
  
*def* StartMain(*calcList*):  
 print("List :",*calcList*)  
 print("Sorted List :",sorted(*calcList*))  
 print("\nMean :",round(statistics.mean(*calcList*),2))  
 print("Median :",statistics.median(*calcList*))  
 *try*:  
 print("Mode :",statistics.mode(*calcList*))  
 *except*:  
 *pass* print("\nUpper Quartile :",QuartileRange(3,*calcList*))  
 print("InterQuartile :",QuartileRange(0,*calcList*))  
 print("Lower Quartile :",QuartileRange(1,*calcList*))  
 print("\nVariance(σ^2) :",round(Variance(*calcList*),2))  
 print("Standard Deviation(σ) :",round(math.sqrt(Variance(*calcList*)),2))  
 print("\nSkewness :",round(3\*(statistics.mean(*calcList*)-statistics.median(*calcList*))/(math.sqrt(Variance(*calcList*))),3))  
 print("\nUpper Outlier Range :",round(QuartileRange(3,*calcList*)+1.5\*QuartileRange(0,*calcList*),2))  
 print("Lower Outlier Range :",round(QuartileRange(1,*calcList*)-1.5\*QuartileRange(0,*calcList*),2))  
  
 Stay = *True  
 while* Stay:  
 choice = input("Options = Coding(C) or Correlator(R) or Percentiles(P) [Nothing to return to main menu] : ")  
 print("\n\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-\n")  
 choice.lower()  
 *if* choice.upper() == "R":  
 Correrlater(*calcList*)  
 *if* choice.upper() == "C":  
 StartMain(FreakingCoding(*calcList*))  
 *if* choice.upper() == "P":  
 print("Point :",PointPercentile(*calcList*))  
 *else*:  
 *if* choice != "":  
 print("\nInput Error ---- [Restarting Choice]")  
 Stay = *False*